HEWLETT PACKARD

User's Guide

Debug64700

Digitized by the Internet Archive
in 2025

https://archive.org/details/manualsplus_05076

Hewlett-Packard
Printed in U.S.A.

To reorder complete document assembly with binder(s), order HP part number:

B1471-99009

To order individual manual within binder(s), order part number shown on title page of manual.

User's Guide

Debug64700

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

© Copyright 1993, 1994, Hewlett-Packard Company.

This document contains proprietary information, which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

HP is a trademark of Hewlett-Packard Company.

Microtec is a registered trademark of Microtec Research Inc.

OSF/Motif and Motif are trademarks of the Open Software Foundation in the U.S. and other countries.

SunOS, SPARCsystem, OpenWindows, and SunView are trademarks of Sun Microsystems, Inc.

UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

Hewlett-Packard
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901-2197, U.S.A.

RESTRICTED RIGHTS LEGEND Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013. Hewlett-Packard Company, 3000 Hanover Street, Palo Alto, CA 94304 U.S.A. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Printing History

New editions are complete revisions of the manual. The date on the title page changes only when a new edition is published.

A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual revisions.

Edition 1	B1471-97010, April 1993
Edition 2	B1471-97011, April 1994

In This Book

This book describes the Debug64700 interface, which connects the HP 64700 Debug Environment to the SoftBench Environment. If you are learning about SoftBench or the HP 64700 Debug Environment, you should explore each of these systems first before studying this manual, which explains how the two systems are integrated together. This book is organized into four parts whose chapters are described below.

Part 1. Quick Start Guide

Chapter 1, Getting Started, presents a step-by-step tutorial on how to perform a few basic tasks using the Debug64700 interface.
Chapter 2, Setting Up Your Project for Debug64700, describes the steps you need to take to set up your project so that it works correctly with the Debug64700 interface.

Part 2. User's Guide

Chapter 3, Controlling the Debug64700 Interface, shows you how to start up and use the Debug64700 interface.
Chapter 4, Debug64700 Interaction with Other SoftBench Tools, describes how the interaction between Debug64700 and other SoftBench tools is set up.

Part 3. Reference

Chapter 5, Debug64700 Application Resources describes the common user-modifiable X resources for the Debug64700 application.
Chapter 6, SoftBench Messages Supported, describes the messages that can be forwarded to the Debug64700 tool to generate SoftBench Messages as well as the SoftBench Messages the Debug64700 tool can receive.
Chapter 7, Common Problems and Solutions, provides information about some of the common problems you may encounter when using the Debug64700 tool.

Part 4. Installation Guide

Chapter 8, Installation, contains information related to the installation and setup of Debug64700.

For More Information ...

- For an overview of the SoftBench System, go to the *SoftBench Overview Manual*. For more specific information on the SoftBench System, go to the manuals referenced in the *SoftBench Overview Manual*.

- For information on your emulator, go to the *Emulator User's Guide Manual* for the particular emulator you are using.

Contents

Part 1 Quick Start Guide

1 Getting Started

Step 1. Start the demo 15
If you have problems starting the demo 20
Step 2. Select and start the HP 64700 system 21
Step 3. Use the Static Analyzer 23
Step 4. Use the Static Analyzer graph to set breakpoints 25
If you could not set a breakpoint using the Static Analyzer 27
Step 5. Invoke the SoftBench Editor 28
Step 6. Rebuild the target executable file 29
Step 7. Use the AxLS Branch Validator 30
Step 8. Use the Software Performance Analyzer (HP 64700 emulation system only) 32

2 Setting Up Your Project for Debug64700

Step 1. Compile your programs for Static Analysis 35
Step 2. Set X resources for the new action keys 36
Step 3. Verify that SoftBench is properly configured 37
Step 4. Start the Debug64700 system 38

Part 2 User's Guide

3 Controlling the Debug64700 Interface

To start the Debug64700 interface 45
To change the target executable (context) 46
To select an HP 64700 system 47
To select the interfaces to start 48
To specify a configuration file 49
To specify interface startup commands or command files 51
To set the load options 53
To view interface status 54
To start additional interfaces 55
To access Debug64700 online help 57

4 Debug64700 Interaction with Other SoftBench Tools

The SoftBench Editors 61
The SoftBench Builder 62
The SoftBench Static Analyzer 65
The HP Branch Validator 66

Part 3 Reference

5 Debug64700 Application Resources

Debug64700*interfacePreselect*String: Emul 70
Debug64700*systemName*String: em68000 71
Debug64700*configFile*String: c68000.EA 71
Debug64700*debugCommand*String: debuggercommand.com 71
Debug64700*emulCommand*String: emululatorcommand 71
Debug64700*perfCommand*String: performancecommand 71
Debug64700*loadOptions*String: Executable 72
Debug64700*loadNotification*String: True 72
Debug64700*autoStart*String: False 72
Debug64700*bbaOptions*String: SHOW-HISTOGRAM 72

6 SoftBench Messages Supported

SoftBench Edit Commands 75
SoftBench Build Commands 76
SoftBench BBA Command 77
SoftBench Generic Message Commands 78
SoftBench Messages Received 80

7 Common Problems and Solutions

Cannot Set Breakpoint from Static Analyzer to Debug Environment 84
Executable File Doesn't Load 85
What Does the Space in the Context Mean? 86
Can I Load Multiple Executables Into the Debug Environment? 86
Why Does the System Ask for a Configuration File? 87
Where Does the Executable File and Configuration File Come From When Running Remote? 87
What is the Advantage of Using the Supplied Compile Scripts to Generate Static Analysis Object Files? 88

Part 4 Installation Guide

8 Installation

Supported Emulator Systems and Software Versions 93
Installing the Debug64700 Software 94
Customization of the SoftBench Environment 95
Updating the MANPATH Environment Variable 97
Using Debug64700 Remotely 98
Adding Magic Strings 100
Installation Verification 101

Index

Part 1

Quick Start Guide

A one-glance overview of the product and a few task instructions to help you get comfortable.

1

Getting Started

Getting Started

This tutorial gives you step-by-step instructions on how to perform a few basic tasks using the Debug64700 interface. It uses the demo program supplied with the Debugger/Emulator, Debugger/Monitor, or Emulator/Analyzer interface software. This tutorial assumes:

- You have the HP AxLS C cross compiler for the HP emulator or ROM monitor you have purchased. If you do not have the HP AxLS cross compiler installed, go to the chapter titled "Setting Up Your Project for Debug64700".
- You have already verified operation of your emulation or ROM monitor system and SoftBench.

The following steps show you many of the advantages of using the Debug64700 system in combination with SoftBench.

This chapter shows you how to:

1. Start the demo.
2. Select and start the HP 64700 system.
3. Use the Static Analyzer.
4. Use the Static Analyzer graph to set breakpoints.
5. Invoke the SoftBench Editor.
6. Rebuild the target executable file.
7. Use the AxLS Branch Validator.
8. Use the Software Performance Analyzer (HP 64700 emulation system only).

After running this demo, go to the next chapter titled "Setting Up Your Project for Debug64700", which explains the steps you need to take to set up your system for the Debug64700-to-SoftBench connection.

Step 1. Start the demo

A demo program and its associated files are provided with the emulator's Graphical User Interface.

1 Change to the demo directory with the **cd $HP64000/demo/debug_env/hpXXXXX** command (where XXXXX is the HP model number for your emulator). For example, if you're using an HP 64742 emulator (for Motorola 68000), enter the command:

```
$ cd $HP64000/demo/debug_env/hp64742 <RETURN>
```

HP64000 is a shell variable which is set to the directory where HP 64000 software has been installed (typically /usr/hp64000).

For the Debugger/Monitor, use the equivalent emulation directory for your processor:

68000	hp64742
68302	hp64746
68020	hp64748
68030	hp64747
68040	hp64783
6833x	hp64749
68340	hp64751
68360	hp64780

2 Start the Debug64700 demo.

```
$ ../debug64700/Startdebug64700 <RETURN>
```

or, to run the demo with the Debugger/Monitor:

```
$ ../debug64700/Startdebug64700 -mon <RETURN>
```

The -mon option compiles the files for the Debugger/Monitor. Refer to the "READMEMON" file in the demo directories to determine how to set up the demo files for your ROM monitor target.

As the Startdebug64700 script runs, the following information is displayed.

```
You are currently in the directory:
/usr/hp64000/demo/debug_env/hp64742

You must run this demo in a writable directory!
Would you like to copy the demo files to a different directory?
[y/n] (y)
```

Type "y" and press <RETURN>, or press <RETURN>.

```
Please enter the directory (full path name) to which you
want the demo files to be copied (default: $HOME/demo).
>
```

Enter a full path directory name, or select the default by pressing <RETURN>.

```
Copying files to $HOME/demo/debug_env/hp64742 ...
Copy succeeded ...
Re-making to update symbol paths and create Static Analysis files ...

/usr/hp64000/bin/cc68000static -I. -LM -OG -c main.c
/usr/hp64000/bin/cc68000static -I. -LM -OG -c init_system.c
/usr/hp64000/bin/cc68000static -I. -LM -OG -c update_sys.c
/usr/hp64000/bin/cc68000static -I. -LM -OG -c proc_spec.c
/usr/hp64000/bin/ld68k -c Linkcom.k -L -o ecs.x main.o init_system.o
update_sys.o proc_spec.o >ecs.MAP
```

At this point the files have been copied to your specified directory and have been recompiled to update the symbol paths. Additionally, static analysis files (for example, main.q), to be used by the Static Analyzer tool, have been created.

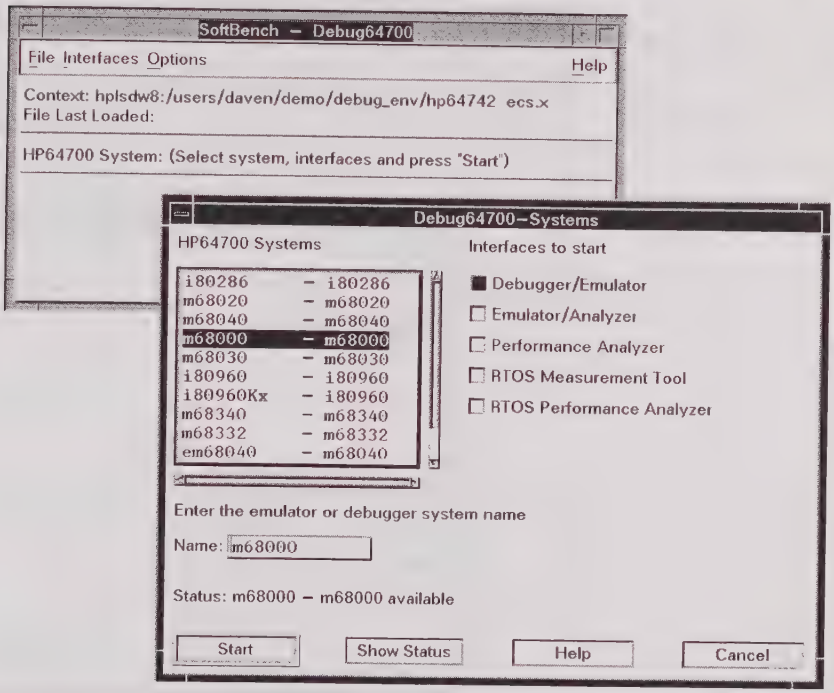
```
Making a local version of softinit in this directory.
Appending the "-demo" option to the DEBUG (debug64700) tool.
Starting SoftBench using this local softinit ...
```

A local version of softinit has been created using the default softinit file (default=/usr/softbench/config/softinit) and it has been modified specifically for this demo. This version of softinit is only to be used with this demo program. Finally, SoftBench is started using this softinit file. If SoftBench is already running, you will see the message below instead of the Starting SoftBench ... message.

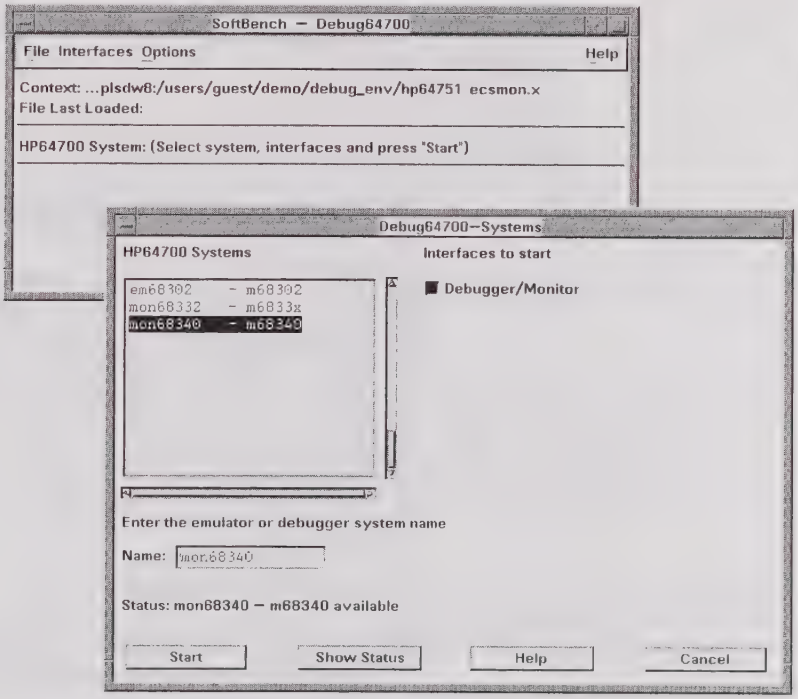
```
SoftBench is being reinitialized to use the local softinit

Starting "debug64700 -demo"
```

This informs you that the script is now starting debug64700 in a demonstration mode of operation. In addition to seeing the normal

SoftBench startup windows you will also see the SoftBench-Debug64700 interface as shown in the following figure.



If you are using the Debugger/Monitor, the SoftBench-Debug64700 interface will look like the following figure.




```
Now you can start and restart demo by either:

  1) Double clicking on "ecs.x" in the Development Manager
  2) Using Tool Manager Start by double clicking "DEBUG"
     after you have changed the context to include the file "ecs.x"
  3) Entering the command "debug64700 -demo" in a terminal window

If you quit SoftBench the demo setup will be lost.
You can restore it by changing directory to your writable demo
directory
and executing either:
        "Startdebug64700"
              or
        "softbench; softmerge -f
$HOME/demo/debug_env/hp64742/softinit"


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
NOTE: You should now issue the following command in this
      terminal window to change to the new demo directory:

      cd $HOME/demo/debug_env/hp64742
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

The above text explains how to restart this demonstration. Don't stop SoftBench until after you've completed the rest of this demonstration.

3 Change to the new demo directory (as listed in the NOTE above). For example:

```
$ cd $HOME/demo/debug_env/hp64742 <RETURN>
```

If you have problems starting the demo

☐ Does your PATH variable contain paths to the /softbench/bin and /hp64700/bin directories?

☐ If the HP 64000 products are not installed in /usr/hp64000, have you exported the shell variable HP64000 and set it to the equivalent path of /usr/hp64000?

Note: On Sun Solaris, HP 64000 products are typically installed in /opt/hp64000.

☐ Can you start SoftBench manually by entering "softbench"?

If you cannot start SoftBench, refer to the *SoftBench Installation* manual.

☐ If SoftBench is not installed in /usr/softbench, have you exported the shell variable SOFTBENCH and set it to the equivalent path of /usr/softbench?

To set shell variables such as HP64000:

If using sh(1) or ksh(1), enter:

```
$ HP64000=/mydir/usr/hp64000 <RETURN>
```

```
$ export HP64000 <RETURN>
```

If using csh(1), enter:

```
$ setenv HP64000 "/mydir/usr/hp64000" <RETURN>
```

Step 2. Select and start the HP 64700 system

After the demo is successfully started (as described in Step 1), a "Debug64700—Systems" dialog box is opened so that you can select and start the HP 64700 system you'll be using.

If you have started this demo with the "-mon" option for the Debugger/Monitor, you must select a Debugger/Monitor system. Only the Debugger/Monitor interface is available to be selected and must be selected for the demo to continue.

1 Select the HP 64700 system either by positioning the mouse pointer over the appropriate system name and clicking the *select* mouse button, or by positioning the mouse pointer in the "Name" field and typing the name of the system.

You must select an HP 64700 system that matches the directory where you started the demo script. For example, if you're using the HP 64742 emulator, select a system whose processor type is "m68000".

If you have no system names to choose from, you must edit the $HP64000/etc/64700tab.net file to add them. You can use the File→Edit HP64700 Systems command in the "SoftBench — Debug64700" window. Refer to your emulator *User's Guide* and the comments in the 64700tab.net file for information about proper contents.

2 After selecting an HP 64700 system, the status is shown in the "Status" field. You can also display the status of the HP 64700 system by clicking on the "Show Status" button.

If you typed a name into the "Name" field, you must click on the "Show Status" button to get valid information about the status of the system.

The "Status" field should show that the system is available.

If a system is not available because it is locked, you can click on the "Unlock" button to make it available. Because unlocking a HP 64700 system may cause another user to lose a valuable measurement, check with the user before you unlock their session.

If a system is unusable (you get the message: HP64700 I/O error), verify that the system has power and that it is connected through the LAN or a serial port.

3 Start the HP 64700 system by clicking on the "Start" button.

It takes a few seconds for the Debugger/Emulator interface to start and the demo configuration file (Configall.EA) executable file (ecs.x) to be loaded.

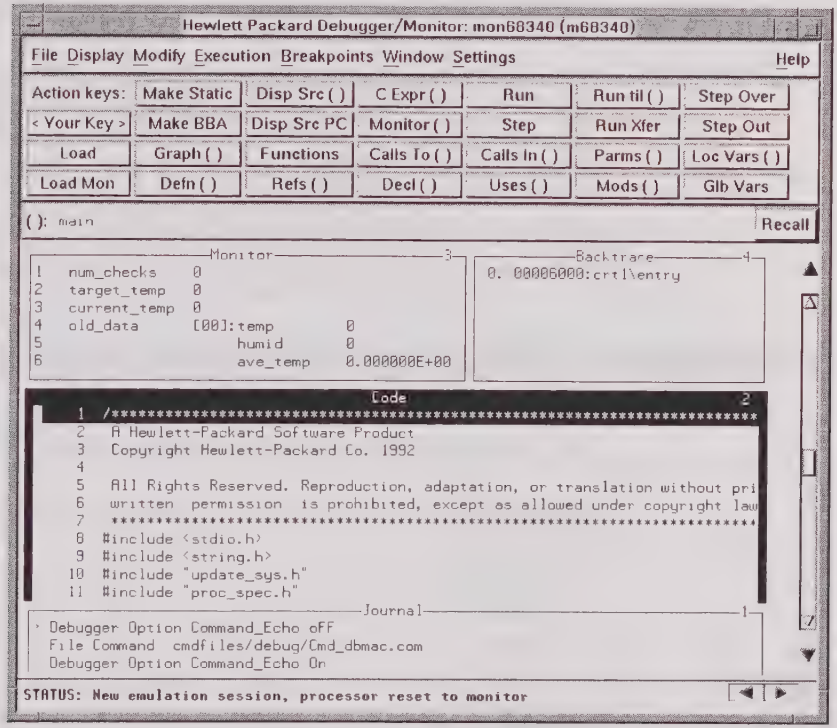
The Debugger/Emulator interface is preselected in the demo program. If you did not purchase this interface, the Debugger/Emulator selection will be half-toned and unselectable. If this is the case, select the Emulator/Analyzer interface instead.

You must select one or the other of these interfaces, or both, to continue this demonstration. It is assumed that you have the Debugger/Emulator interface selected.

To get additional information about starting the Debug64700 system click on the "Help" button.

Step 3. Use the Static Analyzer

If the demo has proceeded correctly to this point, you should see the Debugger/Emulator or Debugger/Monitor interface as shown below.



Notice the four lines of action keys. The top two rows control operations associated with the Debugger, and the bottom two rows demonstrate features of the Debug64700 connection to SoftBench.

You can customize these action keys by modifying the application's X resources. Refer to your Debugger/Emulator and Emulator/Analyzer *User's Guides* for information on customizing action keys.

Wait until the interface is fully initialized (until the half-toned action keys appear normal) before performing the following steps.

1 Click on the Functions action key.

This action key sends a message to the Static Analyzer to display all of the functions in the demonstration program.

Two windows appear: one called Static Analyzer Output, and the other called SoftBench - Static Analyzer. Close the Static Analyzer Output window by clicking on "Close". In the SoftBench - Static Analyzer window, you should see a list of all the functions in the demonstration program (ecs.x).

2 Make sure "main" is in the entry buffer (labeled "():" in the Debugger interface), and click on the Calls In ( ) action key.

This action key sends a message to the Static Analyzer to display all of the function calls inside the function "main".

The list of functions shown in the SoftBench - Static Analyzer window are now just the functions called by the function "main".

3 In the same way, you can click on the action keys labeled:

Calls To ()
Decl ()
Parms ()
Uses ()
Loc Vars ()
Mods ()
Refs ()
Glb Vars

Refer to your *SoftBench Static Analyzer Users Guide* for a complete discussion of the operations associated with these action keys.

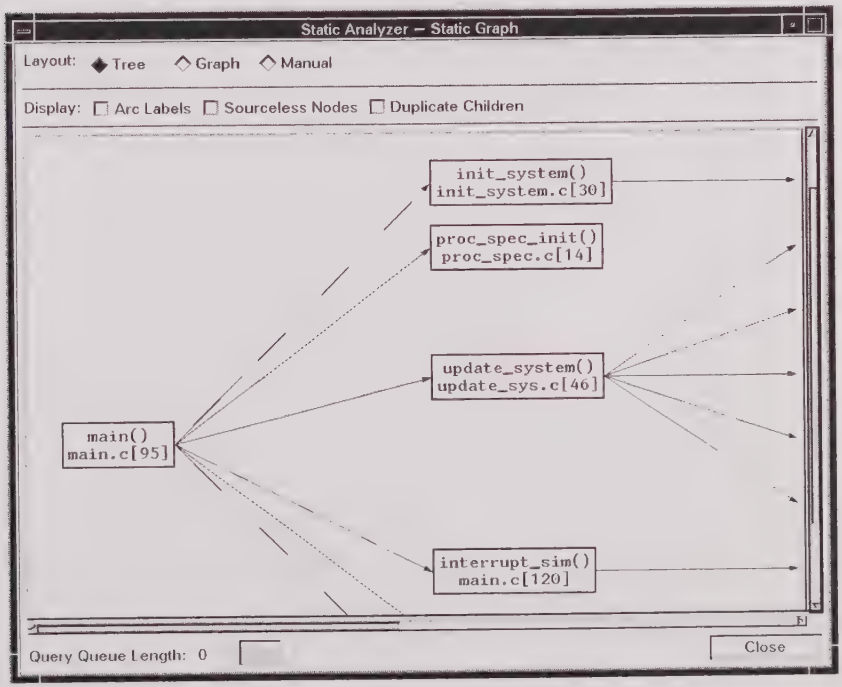
Step 4. Use the Static Analyzer graph to set breakpoints

The Static Analyzer graph is only available in SoftBench version B.00.00 or greater. If you are using an earlier version of SoftBench, skip this step.

1 Make sure "main" is in the entry buffer, and click on the **Graph ()** action key.

2 Select main in the Static Analyzer using the left mouse button; then, access the pop-up menu with the right mouse button and select "Recursive Calls Within" (also with the right mouse button).

A graphical representation of the function call graph is shown.



3 Select the arc pointing to the function update_system using the left mouse button; then, access the pop-up menu with the right mouse button and select "Set Break Point" (also with the right mouse button).

In the Debugger, you will see the breakpoint displayed in the Breakpoint window.

4 To verify that the breakpoint is set, choose the **Execution→Run→from Transfer Address** command in the Debugger window.

Program execution should halt on the line that calls the function update_system.

If you could not set a breakpoint using the Static Analyzer

☐ **Is the Debug64700 interface still running?**

If you exited the interface, the communication between SoftBench and the Debug Environment cannot occur. Choose the File→Exit→Released command from the Debugger window, and restart the demonstration.

☐ **Make sure that "main" is in the entry buffer "():" before you press the "Graph ()" action key.**

☐ **If the breakpoint will not set, make sure the Static Analyzer knows the context of the DEBUG tool (debug64700).**

Pressing any of the action keys to send a Static Analyzer command will set this context.

Step 5. Invoke the SoftBench Editor

1 With the current program counter at the address of the breakpoint (as shown by the highlighted line in the Debugger's Code window), choose the **File→Edit→At PC Location** command.

This command sends a message to the SoftBench Editor that is currently selected as the EDIT tool. This editor may not be your default editor because the demo program is using the default tool configuration file "softinit" supplied with SoftBench.

2 Choose the **File→Quit File** command in the SoftBench Editor window.

You may want to experiment with other edit commands and verify that the SoftBench editor is invoked each time. The Debugger's Breakpoints→Edit/Call Macro command may or may not invoke the SoftBench editor. Newer versions of the Debugger will invoke the SoftBench editor.

Step 6. Rebuild the target executable file

- If you have the HP AxLS compiler installed, or if you have modified the make file to work properly with your compiler (where ecs.x is the target of your makefile), press the **Make Static** the action key.

  This action key changes the dates on the ".c" files and sends a message to the SoftBench Build tool to build the target ecs.x file. (A better software development practice would be to remove the ".o" files or to invoke the "clean" target of the Makefile.)

  When the build completes, the executable file ecs.x file is reloaded into the Debugger, and you are notified of the load.

  Also, note that if the Static Analyzer is active, it rereads the static analysis files.

Step 7. Use the AxLS Branch Validator

If the AxLS Branch Validator product is installed, you can configure your executable for branch analysis. If this product is not installed, go on to the next step.

1 Select the **Make BBA** action key.

This action key changes the dates on the ".c" files and sends a message to the SoftBench Build tool to build the target ecs.bba file (which creates a branch analysis version of the ecs.x executable file). (A better software development practice would be to remove the ".o" files or to invoke the "clean" target of the Makefile.)

The new ecs.x executable file is loaded after the build is complete.

2 After the load of the executable file, choose the Debugger's **File→Store→BBA Data...** command, enter the file bbadump.data, and click on the "OK" button.

First, this command copies the branch analysis data to a file named "bbadump.data". Then, a SoftBench message is sent to the Branch Validator to display a histogram of Branch Coverage using this data file. This first histogram should all be zero's because you have not started running the executable program.

3 Run the executable by choosing the Debugger's **Execution→Run→From Transfer Address** command.

4 After a second or two, break the execution using a <CTRL>c.

On some Debugger/Monitors, you may have to press the target Abort button to break execution.

5 Again, dump the branch analysis data by choosing the Debugger's **File→Store→BBA Data...** command, entering the file name bbadump.data, and clicking on the "OK" button.

Notice that the histogram reflects the percent of branches executed in the various functions of the program.

Step 8. Use the Software Performance Analyzer (HP 64700 emulation system only)

If you have the Software Performance Analyzer installed, you should be able to start the Performance Analyzer interface and directly make performance measurements. If the Software Performance Analyzer is not installed, go to the next chapter.

1 Click on the "Start Additional..." button in the Debug64700 window. Select the "Performance Analyzer" interface, and click on the "Start" button.

2 With the Performance Analyzer interface running and the emulator running the user program, click on the **Func Duration** action key to obtain a Function Duration histogram.

If you are using the Motorola 68040 microprocessor, you must compile your program with the marker preprocessor first. To do this, click on the Make Mark action key to invoke the marker preprocessor before you enter the Performance Analyzer.

Refer to the *Software Performance Analyzer User's Guide* or online help in the Software Performance Analyzer for a complete description of the advantages of using markers.

2

Setting Up Your Project for Debug64700

Setting Up Your Project for Debug64700

This chapter describes the steps you need to take to set up your project so that it works correctly with the Debug64700 interface.

This chapter shows you how to:

1. Compile your programs for Static Analysis.
2. Set X resources for the new action keys.
3. Verify that SoftBench is properly configured.
4. Start the Debug64700 system.

After you have performed these steps, you can select and start an HP 64700 system and use it with SoftBench as was shown in Steps 2 through 8 of the "Getting Started" chapter.

Step 1. Compile your programs for Static Analysis

The Static Analyzer can work in two modes:

- Source File Mode. The mode uses source files and does not require any compiling. This mode is simpler to use at the expense of limited Static Analysis features. In particular, the Static Analysis features of Global Variables, Function Local Variables, Function Parameters, and Variable Modification cannot be shown.
- Object Mode. This mode allows the full features of the Static Analyzer to be used. To enter this mode you must compile your files in a way to create ".q" Static Analysis files.

If you are using the HP AxLS or the MRI C Cross compilers, you can create the ".q" Static Analysis files by invoking a special static compile script called XXXXXXstatic in place of your standard compile command (where XXXXXX is your standard compile command).

If you are not using the HP AxLS or MRI C cross compilers, you will not be able to create these Static Analysis files, and you will have to rely upon Source files. (Note that, as an independent exercise, you could create your own static analysis scripts for your compiler.)

1 Edit the Makefile and change XXXXX to XXXXXstatic.

For example, if you are using the HP AxLS compiler cc68000, replace cc68000 with cc68000static.

2 Touch all of the ".c" files ("touch *c") so they will be recompiled.

3 Finally execute the "make <target>" command to create a new executable file and generate the static analysis files at the same time.

Step 2. Set X resources for the new action keys

Action keys in the Debugger/Emulator, Emulator/Analyzer, and Performance Analyzer interfaces are defined by setting X resources. The Debug64700 demo files include a template for setting up SoftBench-related action keys in these interfaces.

1 Copy the "Xdefaults.template" file to your home directory.

```
$ cp
$HP64000/demo/debug_env/debug64700/Xdefaults.template
$HOME/. <RETURN>
```

2 Edit the "Xdefaults.template" file and replace all occurrences of the string "INSERT_EMULATOR" with the processor name of the emulator that you are using (for example, m68000 for the HP 64742/3/4 emulators).

Remember, the last action key must not have a line continuation character, "\", on the line. (Refer to your emulator manual for a complete description of action key format.)

In addition, you may need to change the Make action keys to properly make your executable. For more complete details, refer to the "The SoftBench Builder" section in the "Debug64700 Interaction with Other SoftBench Tools" chapter.

Save your changes and exit the editor.

3 If the RESOURCE_MANAGER property exists (as is the case with HP VUE — if you're not sure, you can check by entering the **xrdb -query** command), use the **xrdb** command to add the resources to the RESOURCE_MANAGER property. For example:

```
$ xrdb -merge -nocpp $HOME/Xdefaults.template <RETURN>
```

Otherwise, if the RESOURCE_MANAGER property does not exist, append the temporary file to your $HOME/.Xdefaults file. For example:

```
$ cat $HOME/Xdefaults.template >> $HOME/.Xdefaults
<RETURN>
```

Step 3. Verify that SoftBench is properly configured

SoftBench must be properly configured to understand the Debug64700 tool.

1 Verify that the Debug64700 tool is listed in your $HOME/.softinit file.

If not, copy the description of the Debug64700 tool from the system softinit file (/usr/softbench/config/softinit).

The Debug64700 tool must be the last DEBUG tool in your .softinit file (put it at the end of the .softinit file).

2 Start SoftBench in the directory where your demo is located. You can start SoftBench by entering the **softbench** command.

Step 4. Start the Debug64700 system

- **Double-click on the executable file list in the DM.**

  For example, if you are using the Debug Environment demo, you can double-click on ecs.x to start the Debug64700 system.

  If the type of executable file does not say "Target Executable", you must use the command line alternate method (described below) to start Debug64700.

  All files ending with ".x" or ".X" are considered to be "Target Executables" for the HP 64700 systems. Make sure you create executable files with ".x" or ".X" extensions when compiling programs for the HP 64700 systems.

- **Or, enter the debug64700 -file <executable_file> command.**

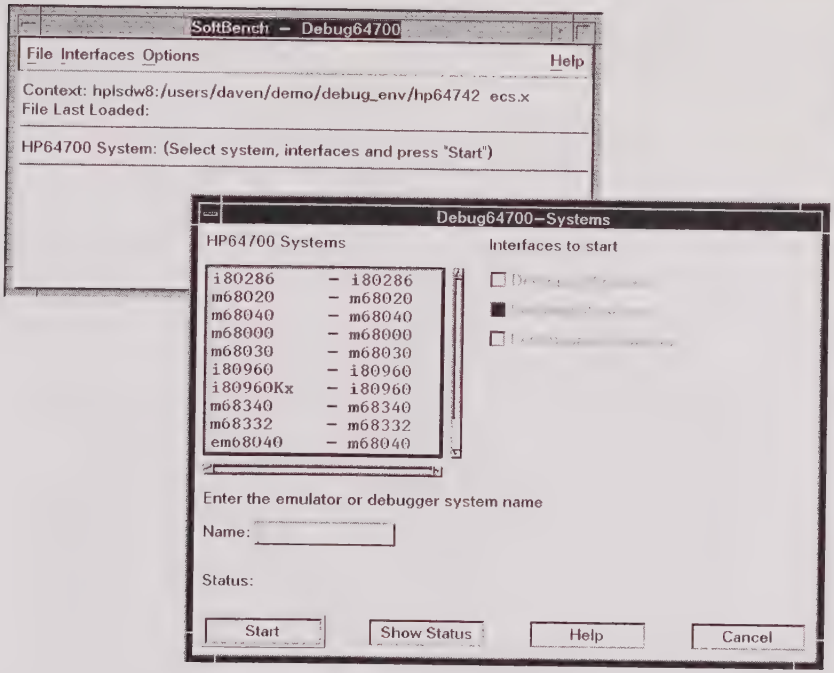
  For example, if you are using the Debug Environment demo, enter:

  ```
  $ debug64700 -file ecs.x <RETURN>
  ```

- **Or, choose the Tool→Start command from the SoftBench Tool Manager window, and select and start the "DEBUG" tool using the Start dialog box.**

  Make sure you set the context to the directory and the file to the appropriate values for your executable.

  Once the Debug64700 system is started, you should see start up windows similar to those shown in the following figure.



Refer to the man page debug64700.1 for additional details about starting the Debug64700 system.

Part 2

User's Guide

A complete set of task instructions and problem-solving guidelines, with a few basic concepts.

3

Controlling the Debug64700 Interface

Controlling the Debug64700 Interface

This chapter shows you how to:

- Start the Debug64700 interface.
- Change the target executable (context).
- Select an HP 64700 system.
- Select the interfaces to start.
- Specify a configuration file.
- Specify interface startup commands or command files.
- Set the load options.
- View the interface status.
- Start additional interfaces.
- Access Debug64700 online help.

To start the Debug64700 interface

- Enter the **debug64700** command in a terminal window.

  To start the Debug64700 interface from a terminal interface with a Target Executable file selected, enter a command like debug64700 -file ecs.x.

  For a complete list of additional options, enter the debug64700 -help command.

- Double-click on a Target Executable file in the SoftBench Development Manager.

- In addition, the interface can be started by a LOAD SoftBench message sent to the DEBUG tool, provided that the message has a file context that is a Target Executable (a file ending in .x or .X). This Target Executable file then becomes the context of the DEBUG64700 tool and, in addition, becomes the executable file that is loaded into the HP 64700 system.

To change the target executable (context)

- When the Debug64700 tool is started, but before the HP 64700 system is started (before you click on the "Start" button), choose the **File→Set Context...** command in the Debug64700 window.

  The context is the context of the DEBUG64700 tool and also reflects the name of the Target Executable file that is loaded when the HP 64700 system is first started.

  The load options commands control the actual loading of the Target Executable and may actually cause the file to never be loaded.

  If the file portion of the context shown is a number and not a Target Executable file, then a Target Executable file has not been specified and the context will be this directory and the number specified. Also, in this mode no file will be loaded.

  If you enter a file called "NONE" the context will revert to a number and the system will not load a Target Executable file.

To select an HP 64700 system

When the Debug64700 tool is started, the Debug64700—Systems dialog box is opened. You can select an HP 64700 system in the following ways.

- **Single-click on the desired system in the list of displayed systems.**

  This list is a summary of the systems described in the 64700tab.net file.

  If the list of HP 64700 systems is out of date, you can quickly edit the 64700tab.net file by choosing the File→Edit HP64700 Systems.... command. After you edit the file to make any necessary additions or deletions, you can simply close the edit window and the list of systems will be updated.

- **Enter the system name in the "System" box and press return.**

  After selecting a system, the status of the system will be updated and shown on the "Status" line.

  If the system is locked to yourself on the current host, you will be able to connect to it without unlocking it. If the system is locked to another user, you will not be able to connect to it until you unlock it. Because unlocking a HP 64700 system may cause another user to lose a valuable measurement, check with the user before you unlock their session. To unlock a HP 64700 system, click on the "Unlock" button.

  If a system is unusable (HP64700 I/O error), verify that the system has power and that it is connected through the LAN or a serial port.

  To obtain the status of a system, enter the system name in the "System" box and click on the "Status" button.

To select the interfaces to start

When the Debug64700 tool is started, the Debug64700—Systems dialog box is opened. After the HP 64700 system is selected, interfaces that are available to be started become selectable.

If you select a Debugger/Monitor target, only the Debugger/Monitor interface will be selectable.

- **Click on the check boxes for the interfaces you wish to start.**

If the system is locked to yourself, you will be able to start interfaces on the locked system. If the interface is locked to someone else, you will not be able to connect to it.

The Emulator/Analyzer interface is preselected by default. This can be changed by setting application resources (refer to the "Debug64700 Application Resources" chapter).

If you did not purchase the Emulator/Analyzer interface, this selection will be half-toned and unselectable. If this condition exists, select the Debugger/Emulator interface.

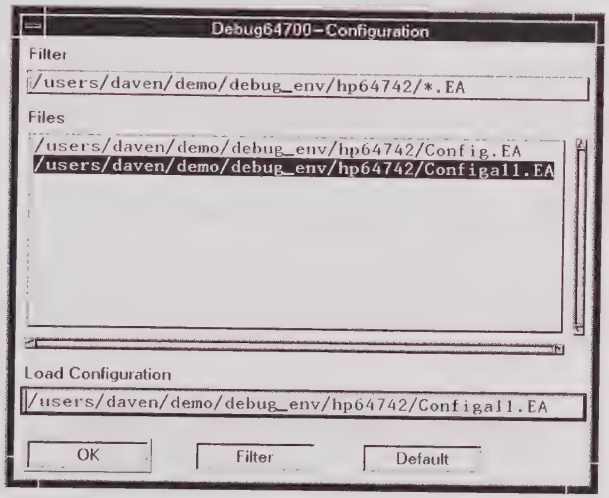
If you have a Software Performance Analyzer, you can also select that interface.

Additionally, if you have the RTOS measurement tool set up you will be able to select the RTOS Measurement Tool and the RTOS Performance Analyzer. You will not be able to select both the Performance Analyzer and the RTOS Performance Analyzer.

To specify a configuration file

1 After the HP 64700 system and the interfaces to be started are selected, you can specify a configuration file by choosing the **Options→Emulator Config...** command. (This menu item is only available before you start the HP 64700 system.)

If you have selected a Target Executable, you will automatically be asked to select an emulator configuration file. A dialog window will appear as shown below.



This dialog box will automatically be opened if you start an interface without specifying a configuration file.

A Debugger/Monitor will only allow you to select configuration files with .MA extensions. When using a Debugger/Monitor, you can typically click on the "Default" button to select the default configuration file. You can also specify an application resource to be DEFAULT and avoid this dialog.

2 Select the line containing your desired configuration file, or type in the name of the configuration file, and click the "OK" button.

This configuration file will be loaded into the HP 64700 system as the interface(s) are started.

If you click the "Default" button, the HP 64700 system will load the default configuration file.

To change the list of shown configuration files to another directory, you can enter the new directory in the filter window and click on the "Filter" button.

Double-clicking on a configuration file will select the configuration file and choose "OK" in one step.

To specify interface startup commands or command files

1 Choose the **Options→Start Commands...** command from the Debug64700 window.

2 In the First_Start_Commands dialog box, enter the commands or command files to be executed as the interfaces start.

Examples

For example, if you have selected the Debugger/Emulator, Emulator/Analyzer, and Performance Analyzer interfaces, you could enter the following commands.

To run the program in the Debugger/Emulator interface, enter the command:

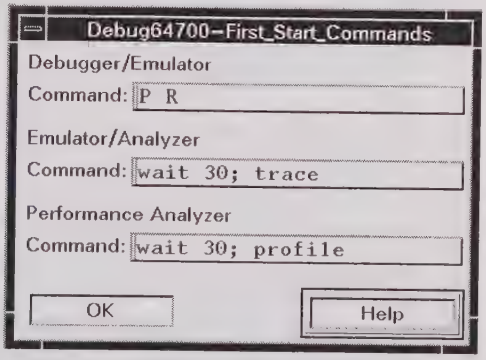
```
P R
```

To start a trace in the Emulator/Analyzer interface, enter the commands:

```
wait 30; trace
```

To start a profile in the Performance Analyzer interface, enter the commands:

```
wait 30; profile
```



Assuming you have selected a Target Executable and an emulation configuration file, these commands cause the following things to happen after clicking on the "Start" button.

- All three interfaces start.
- The Debugger/Emulator interface loads the emulator configuration and the Target Executable.
- The Debugger/Emulator then responds to its "P R" command and starts running the Target Executable.
- The Emulator/Analyzer waits 30 seconds and then executes a trace. The 30 second delay is needed to allow sufficient time for the Debugger/Emulator to initialize and complete the above loading. A delay of more than 30 seconds may be needed if you have a large Target Executable to load, or if you are running on a slower system.
- The Performance Analyzer waits 30 seconds and then executes a profile.

You can also enter the names of command files to execute when the system starts.

Remember, the commands or command files must contain commands in the syntax of the interface to which they apply. The only exception to this rule is the case of a single command file to the Debugger/Emulator. If you enter a command file for the Debugger/Emulator to execute, the Debug64700 interface will check for the existence of the command file and add the needed "F C " to it so that the command file will properly execute in the interface. If you want to run two command files at the start of the Debug64700 interface, you must enter a command like that shown below.

To run a single command file in the Debugger/Emulator:

```
comamndfile1.com
```

To run multiple command files in the Debugger/Emulator:

```
F C comamndfile1.com;; F C commandfile2.com
```

You can also specify application defaults to enter these commands. Refer to the "Debug64700 Application Resources" chapter.

To set the load options

1 Choose the **Options→Load Options...** command from the Debug64700 window.

2 In the Load_Options dialog box, select the appropriate check boxes.

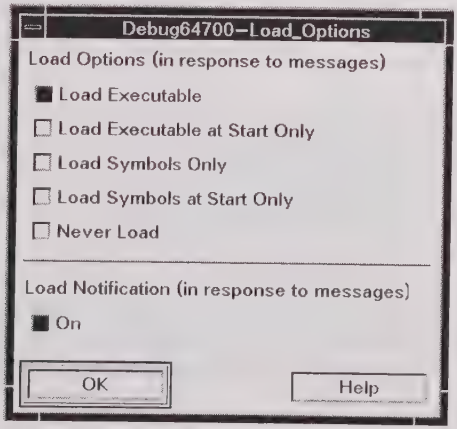
This window allows you to define what happens to the Target Executable in response to system load messages. SoftBench LOAD messages can be generated in a variety of ways, such as double-clicking on a Development Manager Target Executable file, or the completion of a Build, or a command from the Build tool to debug the target. In any case, the response that Debug64700 takes from these LOAD messages is determined in this window.

If you select "Never Load" then all LOAD messages will be ignored.

If you select "Load Executable at Start Only" or "Load Symbols at Start Only", the load will only occur when the HP 64700 system is first started. All LOAD messages after the start will be ignored.

If you select "Load Executable" or "Load Symbols", every load message will generate the requested load.

The "Load Notification" will inform you when a load has completed. Turning this option off will disable this notification.

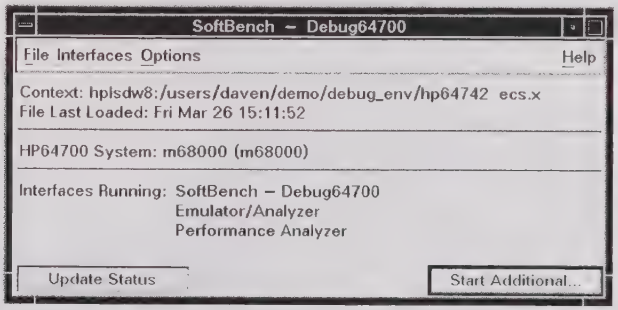


To view interface status

If you exit an interface or start an interface of a different type than what was started initially, the interface status may become out of date. You can get a fresh update of which interfaces are running in the following ways.

- Choose the **Interfaces→Update Status** command from the Debug64700 window.
- Click on the "Update Status" button in the Debug64700 window.

After pressing the "Start" button, the user interfaces requested will take a few seconds to start. The interfaces running will be shown in the Debug64700 window, similar to the window shown below.



The interface status shown cannot distinguish between Emulation/Analyzers and RTOS Measurement Tools, nor can it distinguish between Performance Analyzers or RTOS Performance Analyzers. In addition, the status cannot display the number of Emulation Analyzer interfaces that are started.

The status reflects, instead, whether any of the interfaces are running.

You should not attempt to view the interface status while the interfaces are starting because, as they are initializing, the interfaces will not properly respond to the status request, and you will not see valid status results.

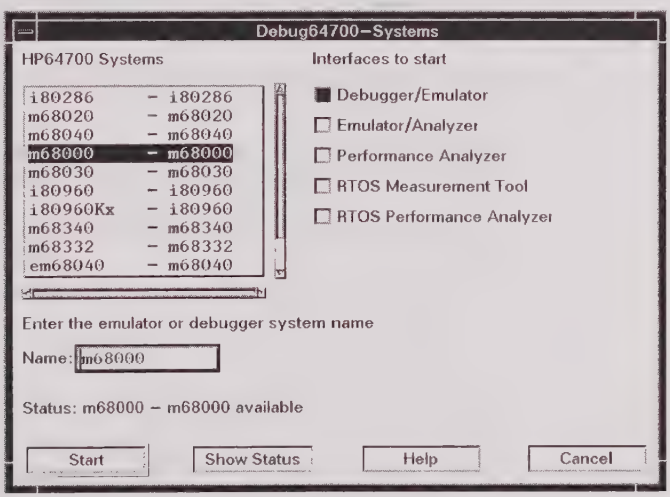
To start additional interfaces

1 Choose the **Interfaces→Start Additional...** command from the Debug64700 window.

Or:

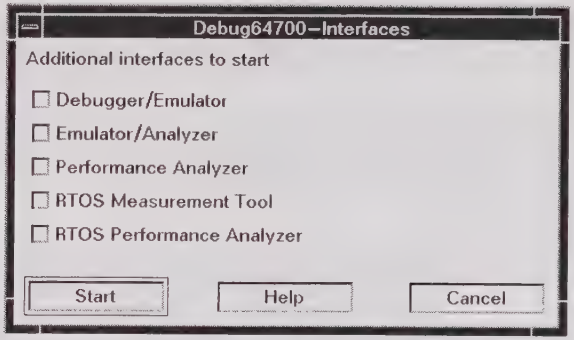
Click on the "Start Additional..." button in the Debug64700 window.

Before starting the HP 64700 system, this command will open the Systems dialog box (shown below) that is used to select an HP 64700 system, interfaces, and start the HP 64700 system.



If you do not select "Start" from this interface, the Debug64700 interface will not complete the connection to SoftBench.

After pressing the "Start" button to start the HP 64700 system, this command will now allow you to select additional interfaces to start. The Interfaces dialog box is shown in the following figure.



2 In the Systems or Interfaces dialog box, select the appropriate check boxes for the additional interfaces you want to start.

3 Simply select the interfaces to start and click on the "Start" button.

Interfaces that are half-toned out are either not available or already running. Additional interfaces of that type cannot be started.

After you click on the "Start" button, allow a few seconds for the interfaces to appear.

To access Debug64700 online help

- Choose the **Help→Application Help** command from the Debug64700 window.
- From any of the SoftBench windows, choose the **Help→List Host Topics** command, and select the "Debug64700 Overview" topic.
- Position the mouse pointer over the Debug64700 window or menu bar and press the F1 key (or "Help" on a Sun keyboard).

For more information on using online help, refer to the "Getting Online Help" chapter in the *SoftBench Overview Manual*.

4

Debug64700 Interaction with Other SoftBench Tools

Debug64700 Interaction with Other SoftBench Tools

This chapter describes the interaction between Debug64700 and other SoftBench tools:

- The SoftBench Editors.
- The SoftBench Builder.
- The SoftBench Static Analyzer.
- The HP Branch Validator.

The SoftBench Editors

The Debug64700 tool will allow Edit commands issued from the Debug Environment to invoke a SoftBench Editor. This is an automatic operation that happens any time the Debug64700 interface is running.

For proper interaction between the Debug64700 tool and the SoftBench Editors, you must only use the TOOL based editors. The COMMAND based editors must NOT be used.

Examples

To invoke the SoftBench editor:

Load a Target Executable file into the Debugger/Emulator or Emulator/Analyzer.

Display the memory at the location of the function "main".

Use the pop-up edit command to edit the file at the line that starts the main routine. This will open the SoftBench Editor.

Now use the pop-up edit command on a different line in the same file. This should only move the cursor in your edit window.

The SoftBench Builder

The Debug64700 tool will submit Build requests to the SoftBench Build tool in response to a Build request from the Debug Environment interfaces.

To simplify the process of requesting builds, an action key that submits the Build request should be defined. An Emulation/Analyzer Build action key may look like this:

```
"Build" "forward bms 'SoftBench_Build_Target ecs.x - NONE
HP64000=$HP64000'"
```

This key requests a Build of the target file ecs.x. In addition, it passes the Compile Mode options of "NONE" and the Build Options of "HP64000=$HP64000" to the Makefile in the current working directory.

Refer to the Xdefaults.template file for additional examples of action keys that can be used to submit Build requests.

When the Build is complete, the executable file will be reloaded as specified by the load options of the Debug64700 interface.

Build Compiler and Assembler Error Messages

The SoftBench Builder has the capability of directly editing the source files where compile errors have occurred. To activate this feature simply double-click on the error in question. Then, use the "Next" and "Previous" buttons in the SoftBench Build window to navigate through your errors.

If you are using the HP AxLS C Cross Compilers or the MRI C or C++ Cross Compilers, this error navigation should work after the Debug64700 product is installed. If you are using a different compiler, you can add the error navigation by editing the builderr.fmt file. Refer to the SoftBench softbuild (1) man page for complete details.

If you are using the HP 64000 or MRI Intel family assemblers (as86, asv20, asv33, asm86) to assemble sources, you may want to improve the format of the error messages by invoking the contributed assembly scripts located in /usr/hp64000/contrib/bin. These scripts modify the assembler error messages into a format that is more suitable for the SoftBench Build tool to handle. Refer to the man page /usr/hp64000/contrib/man/man1/as86err(1) for complete details.

Using the HP AxLS Makefile Templates

Templates to help create Makefiles exist in a couple of different directories.

First, the HP AxLS C cross compilers include templates. These templates are located in the directory /usr/hp64000/lib/mf. This directory contains files CXXXXX.x.p (where XXXXX is the product number of your compiler) which can be used to help build a Makefile. These templates provide a minimal set of features for each of the AxLS C cross compilers.

A second set of templates exists in the /usr/hp64000/inst/debug64700/config directory. This set of templates, although not unique for a specific AxLS compiler, are more complete in providing a much greater number of functions. In order to use these templates with the SoftBench Builder, follow the steps below:

1 Set the following X resources:

```
Build.progTemplateFile: /usr/hp64000/inst/debug64700/config/buildt.p
Build.libTemplateFile: /usr/hp64000/inst/debug64700/config/buildt.l
Build.mkmfDefaultProgName: a.out.x
Build.mkmfDefaultLd: cc68000
```

You may want to copy the templates to a location where you can edit and customize them for your development environment; if you do this, you should modify the X resources settings so they to point to the new template locations.

2 Create a Makefile for the files in the debug environment demo.

First copy the source files, include files, and linker command file of the debug environment demo to a directory of your choosing. Make sure that you do not copy the Makefile.

```
$ cd /usr/hp64000/demo/debug_env/hpXXXXX <RETURN>
```

(Where XXXXX is your emulator product number.)

```
$ cp Linkcom.k *.c *.h /users/daven/examples <RETURN>
```

Also, if you plan on running the built Target Executable, you will want to copy the configuration file Configall.EA to this directory.

```
$ cp Configall.EA /users/daven/examples <RETURN>
```

3 Start the Build tool in the directory that you have chosen.

After the Build tool initializes, choose the Makefile→Create Program... command. In the dialog window that appears, set the appropriate values for the PROGRAM and LD shell variables.

```
PROGRAM=ecs.x
LD=cc68000
```

If you are using a different AxLS C compiler than the 68000 compiler, specify LD=ccXXXXX and in the Add'l Options field CC=ccXXXXX (where XXXXX is your AxLS C cross compiler). Now click on the "OK" button.

4 Start the Build by clicking on the "Build" button to see if you have successfully created a new Makefile.

Change the target to "help" and again click on "Build" to get a listing of the available targets that can be entered.

5 As a last test, start up your emulator and verify that the program runs.

Use the command "debug64700 -file ecs.x," or double-click on the Development Manager file ecs.x. Do not use the "-demo" option, because not all of the demo files needed by the demo have been copied into this directory.

Note: If you were previously running the the Debug64700 demo (Startdebug64700), you must exit SoftBench and restart it so the "-demo" option does not impact this test.

The SoftBench Static Analyzer

The SoftBench Static Analyzer adds a variety of additional capabilities to the Debug Environment. To completely make use of these additional capabilities, you need to compile your files specifically for Static Analysis and add action keys to the Debug Environment interfaces.

Compiling your files for the SoftBench Static Analyzer is accomplished by using a SoftBench C scanning tool called "cscan" tool. This tool creates static analysis (".q") files from your source files.

HP AxLS C and MRI C and C++ wrapper compile scripts have been generated to simplify the process of invoking the cscan tool. To invoke these scripts, simply invoke your HP AxLS cross compiler with ccXXXXXstatic instead of ccXXXXX. In the same way, you can invoke your MRI mcc68k, mcc86, mcc960 or ccc68k compiler with mccXXXstatic to generate static analysis files (.q) for use with the Static Analyzer. Refer to the man pages ccXXXXXstatic.(1), mccXXXstatic.(1) and ccc68kstatic (1) for complete details.

If you are using a compiler different than the ones listed, you can still use the Source Mode of the Static Analyzer and obtain most of the static analysis information.

The Debug64700 tool will submit Static Analysis messages to the SoftBench Static Analyzer in response to Debug Environment interface requests for static analysis. The easiest way to invoke the Static Analyzer from the Debug Environment is to create action keys that submit the Static Analyzer requests. Example static analyzer requests for the Emulator/Analyzer appear like this:

```
"Functions" "forward bms 'SoftBench_Message 5 Request STATIC
SHOW-FUNCTIONS * NOT_QUIET'"

"Calls In ()" "forward bms 'SoftBench_Message 6 Request STATIC
SHOW-CALLS-WITHIN * NOT_QUIET ()'"

"Graph ()" "forward bms 'SoftBench_Message 8 Request STATIC
GRAPH-TOKEN * () - - -'"
```

In addition to sending the Static Analyzer request, the Debug64700 tool will transmit a SET-DEBUG-CONTEXT message informing the static analyzer of the context from which the DEBUG tool is running. This allows the Static Analyzer, in return, to send set breakpoint requests to the DEBUG tool, which in this case is the HP 64700 debug environment.

The HP Branch Validator

If you have purchased the HP Branch Validator, unloading the Branch Validator data from your Debugger/Emulator or Emulator/Analyzer will automatically invoke the HP Branch Validator.

Two messages will be sent out with each unload. The first message informs the HP Branch Validator of the name of the file where the Branch Analysis information was dumped. The second message instructs the Branch Validator interface to update its display.

The second message can be controlled by an application resource that instructs the HP Branch Validator to issue a particular display command. Options include showing a histogram, a file summary, or just the branch coverage results. Refer to the "Debug64700 Application Resources" chapter for more information.

Part 3

Reference

Descriptions of the product in a dictionary or encyclopedia format.

5

Debug64700 Application Resources

Debug64700 Application Resources

You can preconfigure the Debug64700 interface and its interaction with SoftBench by setting the application's X resources.

To change these application resources, copy them from the /usr/hp64000/lib/X11/app-defaults/Debug64700 file to your home directory, modify them, and set them the same way that X resources for actions keys were set in the "Setting Up Your Project for Debug64700" chapter.

This chapter describes the most common user-modifiable X resources for the Debug64700 application:

- Debug64700*interfacePreselect*String:
- Debug64700*systemName*String:
- Debug64700*configFile*String:
- Debug64700*debugCommand*String:
- Debug64700*emulCommand*String:
- Debug64700*perfCommand*String:
- Debug64700*loadOptions*String:
- Debug64700*loadNotification*String:
- Debug64700*autoStart*String:
- Debug64700*bbaOptions*String:

Debug64700*interfacePreselect*String: Emul

This resource preselects the Emulator/Analyzer interface. By specifying one or more of: Debug, Emul, Perf, RTOSMeas, RTOSPerf, you can preselect the Debugger/Emulator, the Emulator/Analyzer, the Performance Analyzer, the RTOS Measurement Tool, and the RTOS Performance Analyzer, respectively. You can preselect two or more interfaces by specifying them together on this application default (...String: Emul Debug).

Debug64700*systemName*String: em68000

This resource defines the system name uniquely identifying an HP 64700 emulator configured in the /usr/hp64000/etc/64700tab.net file.

Debug64700*configFile*String: c68000.EA

This resource specifies the configuration file to be loaded when the HP 64700 system is started. If the value specified is "DEFAULT," the default configuration file for the HP 64700 system will be loaded. If this resource is not defined, you will be prompted to supply a configuration file.

Debug64700*debugCommand*String: debuggercommand.com

This resource specifies the command files or commands to be sent to the Debugger at the initial or first start of the HP 64700 system. The syntax of the commands or command files must conform to the syntax of the Debugger. A command file will be converted to "F C debuggercommand.com" to properly match the required syntax of the Debugger.

Debug64700*emulCommand*String: emululatorcommand

This resource specifies the command files or commands to be sent to the Emulator/Analyzer at the initial or first start of the HP 64700 system. The syntax of the commands or command files must conform to the syntax of the Emulator/Analyzer.

Debug64700*perfCommand*String: performancecommand

This resource specifies the command files or commands to be sent to the Performance Analyzer at the initial or first start of the HP 64700 system. The syntax of the commands or command files must conform to the syntax of the Performance Analyzer.

Debug64700*loadOptions*String: Executable

This resource specifies the HP 64700 load operation to be performed in response to SoftBench LOAD messages. By default the last loaded Target Executable is reloaded in response to a load message. Other options include Executable_start, Symbols, Symbols_start, or Never. The Executable_start and Symbols_start options will only load at the first start of the HP 64700 system. The Never option will cause the HP 64700 to ignore SoftBench LOAD messages.

Debug64700*loadNotification*String: True

This resource specifies that a load notification dialog should appear in response to a SoftBench LOAD message which has caused the HP 64700 system to load an executable or symbol file.

Debug64700*autoStart*String: False

The auto start resource, if set to "True," will cause the Debug64700 interface to use the predefined system and interfaces and to start the interfaces immediately. The "Start" button does not need to be clicked to make the connection. You should have the interfaces, configuration file, and system name predefined (and verify that they work correctly) before you set this option.

Debug64700*bbaOptions*String: SHOW-HISTOGRAM

This resource will specify the type of Branch Validator display to present after you execute a File→Store→BBA Data command. Available options include SHOW-HISTOGRAM, SHOW-SUMMARY, SHOW-RESULTS-ONLY. You must have the HP Branch Validator installed before this resource will become operational.

6

SoftBench Messages Supported

SoftBench Messages Supported

The Debug64700 interface can send out a variety of messages to the SoftBench environment. These messages are sent in response to forwarded commands from the Debug Environment interfaces (Debugger/Emulator, Emulator/Analyzer, and Performance Analyzer) to the Debug64700 tool. The forwarded commands are then translated into the appropriate SoftBench messages.

For example, the command 'forward bms "SoftBench_Edit_File abc.c"', when entered on the command line of a Debug Environment interface, will be translated by Debug64700 into a SoftBench Message like this: "Request EDIT WINDOW current_host current_directory abc.c".

The commands discussed in the following sections can be forwarded to the Debug64700 tool to generate SoftBench Messages.

The Debug64700 interface can also receive a variety of SoftBench messages. These messages are discussed in the section titled "SoftBench Messages Received."

SoftBench Edit Commands

SoftBench_Edit
SoftBench_Edit_File <file>
SoftBench_Edit_File_Line <file> <line number>

Each of the above Edit commands will issue a EDIT WINDOW SoftBench message. <file> is the specified file to edit, and <line number> is the line number to position the cursor. Without a file specified, a file called new.# will be opened in the current working directory. If a full path filename is entered, the directory context will be properly specified. If the full path file name starts with the string "/nfs/", a host context will also be set. If just the basename of the file is passed, the current working directory will become the host and directory context for the file.

Example: forward bms "SoftBench_Edit_File_Line /tmp/junk 35"

SoftBench Build Commands

SoftBench_Build
SoftBench_Build_Target <target>

Each of the above Build commands will issue a BUILD BUILD-TARGET SoftBench message. <target> is the specified target file to build. The current working directory will become the host and directory context where the build is to occur.

After the Build is complete, the Debug64700 tool, in response to the Build complete message, will reload the currently loaded absolute or symbol file into the Debug Environment. If the Debugger/Emulator interface is running, the load command will be passed to that interface. If the Debugger/Emulator interface is not running, the load command will be passed to the Emulator/Analyzer interface. If neither interface is running, an error message will be issued.

Example: forward bms "SoftBench_Build"

SoftBench BBA Command

SoftBench_BBA <file>

The SoftBench_BBA command is used to start up the HP Branch Validator. The SoftBench_BBA command will issue two SoftBench messages. The first message sets the Branch Validator dump file to <file>. The second message requests that a Branch Validator Histogram be displayed.

Example: forward bms "SoftBench_BBA bbadump.data"

SoftBench Generic Message Commands

SoftBench_Message 3 <type> <class> <command>
SoftBench_Message 4 <type> <class> <command><file>
SoftBench_Message 5 <type> <class> <command> <file> <data>
SoftBench_Message 6 <type> <class> <command> <file> <data> <data>
SoftBench_Message 7 <type> <class> <command> <file> <data> <data> <data>
SoftBench_Message 8 <type> <class> <command> <file> <data> <data> <data> <data>

Each of the above SoftBench_Message commands are generic message commands that can be user-defined to send almost any SoftBench message. Users can generate their favorite SoftBench Messages and place them upon Debug Environment action keys. Then, in response to pressing an action key, a SoftBench Message will be generated. The command that should be used depends upon the number of parameters that will be passed.

A SoftBench_Message 3 instructs Debug64700 to make a message consisting only of <type>, <class>, and <command>. The <type> field must be one of "Request", "Notify", or "Failure". The <class> field must be the message class of the SoftBench Tool (for example, EDIT, DM, STATIC ...). The <command> field must be the SoftBench message command for the message class. The <file> field is the specified file to act upon. If a full path filename is entered, the directory context will be properly specified. If the full path file name starts with the string "/nfs/", a host context will also be set. If just the basename of the file is passed, the current working directory will become the host and directory context for the file. The <data> fields are the data items to be passed to the associated commands.

With this generic command structure almost any SoftBench message can be generated and sent to the SoftBench Message Server. For example, this message command will invoke the Static Analyzer and have it graph the function main.

Example: forward bms "SoftBench_Message 8 Request STATIC GRAPH-TOKEN * main - - -"

The Debug Environment will issue a variety of forwarded messages (if the Debug64700 tool is running) in response to menu selections or pop-up menus. These include editing files and invoking the Branch Validator. Other forwarded commands can be added by the addition of action keys. As a

mechanism for learning the possibilities, examine the Xdefaults.template in the demo directory "/usr/hp64000/demo/debug_env/debug64700". This file shows a variety of Action keys that can be added to your debug environment to add Static Analysis and Build messages.

SoftBench Messages Received

The Debug64700 interface can receive a variety of SoftBench messages and then forward appropriate commands to the Debug Environment interfaces. In order for these messages to be received, the context of the message must match exactly (Host Directory File.x) the current context of the Debug64700 interface. Note that the context must include File.x or File.X; otherwise, the command will invoke the Host Debugger.

Request DEBUG LOAD
Notify BUILD BUILD-TARGET

These messages direct Debug64700 to reload the last loaded absolute. The BUILD-TARGET message is accepted if the host and directory match the current working directory. The file context of this message is ignored.

Request DEBUG LOAD-CONFIGURATION file

This message directs Debug64700 to load a configuration file. File must be the full path of the configuration file.

Request DEBUG BP-SET - file line
Request DEBUG BP-SET-PROC - file procedure

These messages direct Debug64700 to set breakpoints.

Request DEBUG BP-CLEAR - file line
Request DEBUG BP-CLEAR-PROC - file procedure

These messages direct Debug64700 to clear breakpoints.

Request DEBUG RUN
Request DEBUG RUN-FROM-RESET
Request DEBUG RUN-FROM-TRANSFER-ADDRESS
Request DEBUG STEP
Request DEBUG STEP-OVER
Request DEBUG STEP-OUT
Request DEBUG BREAK
Request DEBUG RESET-TO-MONITOR

These messages direct Debug64700 to run, step, break, and reset the debugger or emulator.

Request DEBUG DEBUG-COMMAND-FILE command_file
Request DEBUG EMUL-COMMAND-FILE command_file
Request DEBUG PERF-COMMAND-FILE command_file

These messages direct Debug64700 to send a command file or an individual command to the specified interface. If the specified interface is not running, the command will be ignored.

Request DEBUG EXIT-LOCKED
Request DEBUG EXIT-RELEASED

These messages direct Debug64700 to exit the debug environment session.

7

Common Problems and Solutions

Common Problems and Solutions

This chapter provides information about some of the common problems you may encounter when using the Debug64700 tool:

- Cannot set breakpoint from Static Analyzer to Debug Environment.
- Executable file doesn't load.
- What does the space in the context mean?
- Can I load multiple executables into the Debug Environment?
- Why does the system ask for a configuration file?
- Where does the executable file and configuration file come from when running remote?
- What is the advantage of using the supplied compile scripts to generate Static Analysis object files?

Cannot Set Breakpoint from Static Analyzer to Debug Environment

Problem: The Static Analyzer will not allow me to set a breakpoint to the Debug Environment, why?

Solution: The Static Analyzer needs to know the context of the DEBUG tool (Debug64700) before it can send a breakpoint to it.

The easiest solution is to press an action key in the Debugger/Emulator or the Emulator/Analyzer to send a Static Analyzer command to the Static Analyzer. Sending these commands will send the DEBUG context to the Static Analyzer.

A second method of setting the DEBUG context for the Static Analyzer is to use the command Options->Set Debug Context (in the Static Analyzer). Remember to enter the context of the Debug64700 tool exactly as it is shown in the Debug64700 or the SoftBench Tool Manager Window.

Executable File Doesn't Load

Problem: Double-clicking on a Target Executable file in the Development Manager after the Debug64700 system has started, always tries to start a new Debug64700 session. It doesn't load the executable into the existing Debug64700 as I would expect.

Solution: The Development Manager will issue a DEBUG LOAD request by double clicking a Target Executable file. This request is sent out to a DEBUG tool in the exact same context as the Development Manager (Host, Directory, File.x).

When the Debug64700 tool connects to an HP 64700 system, it starts running in a context of the executable or symbol file that is loaded into the system. If the directory to the executable or symbol file is linked to another directory, the context of the Debug64700 interface may switch to the linked directory. As an example consider the following scenario:

1. The user double-clicks on the file /usr/hp64000/demo/debug_env/hp64742/ecs.x in the Development Manager.
2. The Debug64700 interface starts, and the user starts a Debugger/Emulator.
3. When the Debugger/Emulator loads the file ecs.x, it finds that the true path to it is /disc3.11/hp64000_ln/demo/debug_env/hp64742/ecs.x. The Debug64700 interface switches to this context.
4. Another double-click on the file /usr/hp64000/demo/debug_env/hp64742/ecs.x in the Development Manager starts a new Debug64700 interface (whereas what is expected is that the ecs.x file should be loaded).
5. The solution is to change the context of the Development Manager to the directory /disc3.11/hp64000_ln/demo/debug_env/hp64742. This will force the LOAD message that is generated to match the context of the running Debug64700 system. A double-click now on the ecs.x file will generate the desired load of the ecs.x executable without the start of the new Debug64700 session.

What Does the Space in the Context Mean?

Problem: What does the "space" between the directory and file portions of the context mean? Why can't I split the context in the Debug64700 interface?

Solution: The "space" between the directory and the file indicates a separation between what is considered the directory portion of the context and what is considered the file portion of the context.

Starting the Debug64700 tool with a context of "hosta:/users/daven/demo/debug_env hp64742/ecs.x" will direct the current working directory to be "/users/daven/demo/debug_env" while the file portion of the path is considered "hp64742/ecs.x". This context will remain intact until a different executable file is loaded.

When the new executable file is loaded, the context will revert to a more standard context "host:/directory/directory file.x". Debug64700 is not designed to work in a split context mode, so entering a split context into the Debug64700 tool will not be accepted. The only exception to this is by starting in a split context mode or directing the context of the Debug64700 tool to change by using the SoftBench Tool Manager.

Can I Load Multiple Executables Into the Debug Environment?

Problem: Can I loading multiple executables into the Debug Environment by clicking on the files in the Development Manager?

Solution: This is not possible at this time. The load message generated will be for a specific context which must match exactly the context of a Debug64700 system before a load command is generated.

If you need to load multiple executables into your HP 64700 system, you should define a command file to do the load sequence. This command file could be set by an application resource to cause automatic loading of the executables at all initial starts of the HP 64700 system.

Why Does the System Ask for a Configuration File?

Problem: If continuing a session only connects to the session, and no commands are executed, why does the system ask for a configuration file?

Solution: Any time a system is continued, nothing will be loaded. No configuration files, no executables, no command files are loaded. This has been added as a safety feature to prevent users from accidentally changing their locked or running HP 64700 system.

During the start of the Debug64700 system, the complete status of a system is not fully determined until after the point where a configuration file is asked for. You can ignore this request for a configuration file (click on the Default button). If you click on "Default", the configuration file will not be used, and the current configuration will still be in use.

Where Does the Executable File and Configuration File Come From When Running Remote?

Problem: When I run the Debug64700 system on a remote machine, which machine will the Target Executable be obtained from? Where will the configuration file come from?

Solution: The context of Debug64700 running on a remote machine can become a little difficult to comprehend.

The execution host is the machine where the Debug64700 tool is running.

The data host is shown in the context of the Debug64700 interface and is the location where the Target Executable and the configuration file will be searched for.

The current working directory will be set to the data host through a change directory /nfs/data_host/... command. Therefore, most commands that either don't use a file or use a relative file name will execute relative to the data host. Commands that issue a full path name will execute based upon the full path name. If the full path name contains an /nfs/data_host, the command will work with the data_host. Otherwise, the execution host will be invoked through the command.

As an example, storing a configuration specification into a file named /tmp/junk will put the results into the /tmp directory on the execution host. On the other hand, storing the configuration specification into a file named

junk in the current working directory will store the configuration on the data host in a file named /nfs/data_host/users/daven/example/junk.EA.

The execution host, in addition to supplying the execution cycles, will also be used to find the HP 64700 systems and the application resources. All files needed to make the Debug64700 interface run (without regard to what Target Executable file will be running) must, therefore, be available on the execution host.

What is the Advantage of Using the Supplied Compile Scripts to Generate Static Analysis Object Files?

Problem: What is the advantage of using the supplied compile scripts to generate Static Analysis object files?

Solution: The supplied compile scripts will generate Static Analysis object files (files with suffix .q). These object files allow you to obtain a more complete Static Analysis of the components of a Target Executable.

In particular, using these object files allows you to get the following additional capabilities beyond the capabilities of the Source Mode Static Analysis:

Modifications()

Global Variables

Parameters()

Local Variables()

All queries on variables are supported

Scoping is active

Additional C++ capabilities

Refer to your *SoftBench Program Construction Tools User's Guide* for complete details.

Part 4

Installation Guide

Instructions for installing and configuring the product.

8

Installation

Installation

This chapter contains information about:

- Supported Emulator Systems and Software Versions
- Installing the Debug64700 Software
- Customization of the SoftBench Environment
- Using Debug64700 Remotely
- Adding Magic Strings
- Updating the MANPATH Environment Variable
- Installation Verification

Supported Emulator Systems and Software Versions

The Debug64700 tool will work with the following HP emulators:

HP Product No.	Description	HP Product No.	Description
64742	68000 Emulator	64761	80960Kx Emulator
64746	68302 Emulator	64762	8086 Emulator
64747	68030 Emulator	64763	8088 Emulator
64748	68020 Emulator	64764	80C186 Emulator
64749A	68331/332 ACT Mode Emulator	64765	80C188 Emulator
64749B	68332 Emulator	64767	80C18X Emulator
64749C	68331 Emulator	64780	68360 Emulator
64751	68340 Emulator	64783	68040 Emulator
64760	80960Sx Emulator		

All emulator interfaces must be version C.05.XX or greater before Debug64700 will work. Debug64700 will NOT work with terminal mode emulator interfaces

SoftBench versions A.02.1 or B.00.XX or greater must also be installed and operational before Debug64700 can be used.

If you are using SoftBench version A.02.01, then a few features of this tool will not be available. In particular, the Static Analyzer graph and setting of breakpoints from the Static Analyzer graph will not be supported.

Installing the Debug64700 Software

Debug64700 is automatically installed when you load all of the filesets of the Emulator/Analyzer or the Debugger/Emulator products. If by chance you need to select individual partitions to install, you must select the 64700-SOFTBENCH partition to obtain the Debug64700 product.

On Sun SPARCsystems make sure you install the DEBUG64700 fileset. Refer to the *Software Installation Guide* that comes with your media for complete instructions about installing a fileset from a tape.

When installing the Debug64700 product and SoftBench at the same time, make sure you install SoftBench first. If you install SoftBench last, or install a newer version of SoftBench, you must then rerun (as root) the customize script /system/DEBUG64700/customize on HP 9000 Series 300 and HP 9000 Series 700. On Sun SPARCsystems you must run (as root) the envinstall script /usr/hp64000/bin/envinstall.

Note

Rerun the Debug64700 customize script if SoftBench is reinstalled for any reason.

The customize script expects to find SoftBench installed in /usr/softbench and /usr/local/softbench. If SoftBench is installed in a different location, you must export the shell variables SOFTBENCH and LOCALSOFTBENCH before you start the install process. On Sun SPARCsystems, set these shell variables before running envinstall.

After installing the Debug64700 product, you will need to exit SoftBench and the HP Vue environment if they are currently running. The Debug64700 product supplies additional filetypes to the Broadcast Message Server. The Broadcast Message Server will not recognize the new types until it is restarted (by exiting SoftBench and Vue).

Customization of the SoftBench Environment

The Debug64700 product will automatically do a variety of operations to the SoftBench Environment as part of the customization process. You may need to take additional action to complete the customize process for the individual users of SoftBench.

- ☐ The /usr/softbench/config/softinit file is rebuilt to include Debug64700. In the process of creating the softinit file, a copy of the original softinit file is moved to /usr/softbench/debug64700/softinit.old. If for any reason you need to restore the original softinit file, make sure that you include the definition of the Debug64700 tool. You will also need to modify individual users' .softinit files to include the Debug64700 tool if they will be using Debug64700.

Note

The definition of the Debug64700 DEBUG tool must be the last DEBUG tool in your system /usr/softbench/config/softinit and $HOME/.softinit files.

You can find the information about the Debug64700 tool that is needed in the softinit file in the file /usr/softbench/config/softinitsrc/class-defaults/debug64700init. The description of the Debug64700 tool must be on one line.

- ☐ As you customize your $HOME/.softinit file, remember that you should only use the TOOL type EDIT tools. The COMMAND type EDIT tools should not be used because in many instances Debug64700 is relying upon messages that these tools do not provide.

- ☐ The list of filetypes has been updated to add additional types for the Debug Environment. A copy of the original filetypes is moved to the file /usr/softbench/debug64700/C.softtypes.old. If HP Vue was running during the install process it must be restarted on every station that will be using Debug64700 to make it recognize the new filetypes.

- ☐ The builderr.fmt file has been updated to accept HP AxLS C Cross Compiler error messages. The original builderr.fmt has been moved to /usr/softbench/config/builderr.fmt.old.

☐ Various inconsequential files have been added to the SoftBench directories:

Debug64700 application resources have been linked into the /usr/softbench/app-defaults directory.

Action keys have been added to the Development Manager for the Emul Absolute files

Help messages have been added to the appropriate directories for Debug64700 help.

Updating the MANPATH Environment Variable

The MANPATH environment variable should include /usr/hp64000/man to get HP64000 man pages. An example MANPATH may look like this:

```
MANPATH=/usr/man:/usr/contrib/man:/usr/local/man:/usr/hp6
4000/man
```

Using Debug64700 Remotely

The Debug64700 tool can be set up to run remotely. This is where you may want the execution of the Debug64700 tool to operate on a different machine than the one where you are sitting. In setting up to run the Debug64700 tool remotely, consider these points:

- ☐ You must configure your SoftBench system to operate remotely. Can you direct a remote machine to run its editor on a file on your machine? Refer to your SoftBench Installation and make sure your system is configured to operate in this mode.

- ☐ Have you installed the Debug64700 Software on the remote machine? Have you verified that it operates properly on the remote machine?

- ☐ Change your $HOME/.softinit file on the machine that you are sitting at to use the remote machine instead of the local machine like this:

```
DEBUG TOOL FILE TARGET_EXECUTABLE remote_machine debug64700 ...
```

- ☐ You may need to make a .softenv file in your $HOME directory on your remote machine. Refer to the *Installing SoftBench Products User's Guide* for additional information.

- ☐ Make a .Xdefaults or .Xdefaults-<remote machine> file in your $HOME directory on your remote machine. In this Xdefaults file you will want to include any action keys that you want to have appear with the Debug64700 interface.

Once started in a given context, the Debug64700 system will continue to use the host and directory context for all of the files needed (configuration files, symbol files, command files, etc.). This operation will continue until the directory is changed from within the Debug64700 interfaces. When the directory is changed, the Debug64700 interfaces will operate out of the current working directory. When operating remotely, care must be taken not to change the directory away from the host from where you want to obtain your files.

Examples

As an example, if you are running remotely on a machine named johnsys, the machine that you are looking at is named yoursys, and if the absolute file that you want to work with is file.x in the directory $HOME/example of yoursys, the following operation will occur:

Your execution host will be johnsys. Your Debug64700 context will be /nfs/yoursys/$HOME/example/file.x. The Debug64700 tool will search for configuration files in /nfs/yoursys/$HOME/example. Your current working directory will be /nfs/yoursys/$HOME/example.

If you now change directory to $HOME, you have now changed back to the machine named johnsys. The /nfs/yoursys/ must be the first part of the directory that you want to change to /nfs/yoursys/$HOME.

Adding Magic Strings

If you want to improve the capabilities of the file(1) command in determining the types of files, you can add more magic strings to the /etc/magic file. To do this, append the contents of the file /usr/hp64000/inst/debug64700/etc/magic to the end of the /etc/magic file. You must do this as root.

```
# cat $HP64000/inst/debug64700/etc/magic >> /etc/magic
<RETURN>
```

Now try to get the type of the file ecs.x in the debug environment directory. Enter these commands:

```
$ cd $HP64000/demo/debug_env/hp64742 <RETURN>

$ file ecs.x <RETURN>
```

The response will be like this:

```
ecs.x: IEEE-695 object module - for Motorola 68000
```

Installation Verification

To verify installation you, should check that the SoftBench customization listed above has occurred. Once this is verified, you should be able to run the demo in the "Getting Started" chapter.

Index

! $HOME/.softinit file, 37

A action key, Calls In (), 24
action key, Calls To (), 24
action key, Decl (), 24
action key, Functions, 24
action key, Glb Vars, 24
action key, Loc Vars (), 24
action key, Make BBA, 30
action key, Make Static, 29
action key, Mods (), 24
action key, Parms (), 24
action key, Refs (), 24
action key, Uses (), 24
action keys, 23
adding magic strings, 100
application defaults, 70
application resources, Debug64700, 70
assembler error messages, 62
AxLS C cross compiler, 14
AxLS C Cross compilers, 35
AxLS Makefile Templates, using, 63

B BBA command, SoftBench, 77
branch validator, using, 30-31, 66
breakpoints, set by Static Analyzer graph, 25-26
build commands, SoftBench, 76
build compiler error messages, 62
build tool, SoftBench, 29

C C Cross compilers, AxLS, 35
C Cross compilers, MRI, 35
Calls In () action key, 24
Calls To () action key, 24
cannot set breakpoint, 84
changing the Context, 46

changing the Target Executable, 46
common problems and solutions, 84
compile programs for static analysis, 35
compile script, static, 35
configuration file, specifying, 49-50
configuration file, why does the system ask for it?, 84, 87
Context, changing, 46
cross compiler, AxLS C, 14
customization of the SoftBench environment, 95-96

D debug startup dialog, 16
debug64700 -help, 45
Debug64700 application resources, 70
Debug64700 interface and the SoftBench interaction, preconfiguring, 70
Debug64700 interface, starting, 45
Debug64700 system, starting, 38-39
Debug64700, using remotely, 98-99
Debugger/Emulator interface, preselected for demo, 22
Decl () action key, 24
defaults, application, 70
demo program, restarting, 19

E edit commands, SoftBench, 75
Emulator Config..., 49
emulator systems, supported, 93
error messages, assembler, 62
error messages, build compiler, 62
executable file doesn't load, 84-85
executable file, rebuilding, 29

F File: Edit HP64700 Systems..., 47
File: Set Context... command, 46
files, where do they come from when running remote?, 87
Filter, 50
Functions action key, 24

G generic message commands, SoftBench, 78-79
getting started, 13-32
getting started without the demo, 34
Glb Vars action key, 24

H
help, debug64700, 45
histogram, 30
HP 64700 system, starting, 22
HP64700 I/O error, 22
HP64700 system, selecting, 47

I
installation, 92
installation verification, 101
installing the Debug64700 software, 94
interface status, 54
Interfaces: Start Additional..., 55
Interfaces: Update Status, 54

L
Load Executable, 53
Load Executable at Start Only, 53
Load Notification, 53
load options commands, 46
Load Symbols, 53
Load Symbols at Start Only, 53
Loc Vars () action key, 24

M
magic strings, adding, 100
Make BBA action key, 30
Make Static action key, 29
message, HP64700 I/O error, 22
Mods () action key, 24
MRI C Cross compilers, 35
multiple command files to Debugger/Emulator, 52
multiple executables, can I load?, 84, 86

N
Never Load, 53

O
Options: Load Options..., 53
Options: Start Commands..., 51

P
Parms () action key, 24
preconfigure the Debug64700 interface and the SoftBench interaction, 70
problems and solutions, common, 84

R
rebuild the target executable file, 29
Refs () action key, 24
RTOS measurement tool, 48
RTOS performance analyzer, 48

S
selecting an HP64700 system, 47
selecting the interfaces to start, 48
setting the load options, 53
single command file to Debugger/Emulator, 52
SoftBench build commands, 76
SoftBench build tool, 29
SoftBench builder, using, 62-64
SoftBench edit commands, 75
SoftBench editor, invoking, 28
SoftBench editors, using, 61
SoftBench environment, customizing, 95-96
SoftBench generic message commands, 78-79
SoftBench messages supported, 74
SoftBench Static Analyzer, using, 65
SoftBench, starting, 37
SoftBench_BBA command, 77
software version requirements, 93
solutions to common problems, 84
space in the context, 84, 86
specifying a configuration file, 49-50
Start Additional..." button, 55
Startdebug64700 demo, running, 15-19
starting additional interfaces, 55-56
starting SoftBench, 37
starting the Debug64700 interface, 45
starting the Debug64700 system, 38-39
static analysis modes, 35
static analyzer, 29
Static Analyzer graph, use to set breakpoints, 25-26
Static Analyzer, using, 23-24
static compile script, 35
supported emulator systems, 93

T
target executable file, rebuilding, 29
Target Executable, changing, 46

U
Unlock button, 47
Update Status button, 54
Uses () action key, 24
using Debug64700 remotely, 98-99
using the HP Branch Validator, 66
using the SoftBench builder, 62-64

using the SoftBench editors, 61
using the SoftBench Static Analyzer, 65

V versions, software, 93

W what is the advantage of using supplied compile scripts?, 84, 88
where do files come from when running remote?, 84, 87

X Xdefaults.template file, 62

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Hewlett-Packard
Printed in the USA

HEWLETT PACKARD

User's Guide

Simulated I/O

HP 64700 Operating Environment

Simulated I/O

User's Guide

HP Part No. B1471-97009
Printed in U.S.A.
November 1992

Edition 3

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

© Copyright 1991, 1992, Hewlett-Packard Company.

This document contains proprietary information, which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and in other countries.

Hewlett-Packard
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901-2197, U.S.A.

RESTRICTED RIGHTS LEGEND Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in subparagraph (C) (1) (ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013. Hewlett-Packard Company, 3000 Hanover Street, Palo Alto, CA 94304. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Printing History

New editions are complete revisions of the manual. The date on the title page changes only when a new edition is published.

A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual revisions.

Edition 1	B1471-97003, July 1991
Edition 2	B1471-97006, March 1992
Edition 3	**B1471-97009, November 1992**

In This Book

This *Simulated I/O User's Guide* contains five chapters: an introduction to simulated I/O, two reference chapters, and one examples chapter. Also, there is a chapter which describes the error codes used with simulated I/O. A description of the five chapters is given below.

Chapter 1 **Introducing Simulated I/O**. This chapter introduces you to the simulated I/O emulation feature by explaining what simulated I/O is, how it works, what steps you must take in order to use simulated I/O.

Chapter 2 **Configuring Simulated I/O**. This chapter describes the simulated I/O configuration questions which appear as a part of the general emulation configuration questions. This chapter also lists restrictions on the use of simulated I/O.

Chapter 3 **Simulated I/O Protocol**. This chapter describes how emulation system programs must communicate with the simulated I/O process in order to open, close, read from, and write to simulated I/O files/devices. This chapter also describes the communication process whereby emulation systems can execute UNIX processes.

Chapter 4 **Examples**. This chapter shows you how to use the simulated I/O commands to use your workstation's display and keyboard as simulated I/O devices. This chapter also steps you through the process of compiling, linking, and running a simulated I/O demo program.

Chapter 5 **Error Codes**. This chapter describes the simulated I/O error codes as they relate to the particular simulated I/O commands.

Conventions

Example commands throughout the manual use the following conventions:

bold	Commands, options, and parts of command syntax.
bold italic	Commands, options, and parts of command syntax which may be entered by pressing softkeys.
normal	User specified parts of a command.
$	Represents the UNIX prompt. Commands which follow the "$" are entered at the UNIX prompt.
<RETURN>	The carriage return key.

Contents

1 Introducing Simulated I/O

Overview 9
Simulated I/O Is 9
How Does Simulated I/O Work? 9
User Program Request: 10
Response: 11
Using Simulated I/O 11

2 Configuring Simulated I/O

Overview 13
Introduction 13
Answering The Simulated I/O Configuration Questions 14
Restrictions On Simulated I/O 17

3 Simulated I/O Protocol

Overview 19
Introduction 19
Open File (90H) 21
Open File Option 21
Length of Path Name 23
Path Name 23
File Descriptor 23
Close File (91H) 24
File Descriptor 24
Read from File (92H) 25
File Descriptor 26
Number of Bytes to Read 26
Actual Number of Bytes Read 26
Actual Bytes Read 26
Write to File (93H) 27
File Descriptor 27
Number of Bytes to Write 28
Bytes to be Written 28
Actual Number of Bytes Written 28

Delete a File (94H) 29
Path Name Length 29
Path Name 29
Position File Pointer (95H) 30
Relative Byte Offset 31
Starting Location 31
Absolute Byte Offset 32
Position Cursor on Display (96H) 33
File Descriptor (Returned from Open) 33
Line Number 33
Column Number 33
Clear Display (97H) 34
File Descriptor (Returned from Open) 34
UNIX Command (98H) 35
Open Pipe Specification 36
Command Name Length 36
Command Name 36
Simulated I/O Process ID 36
Stdin File Descriptor 36
Stdout File Descriptor 37
Stderr File Descriptor 37
Kill Simulated I/O Process (99H) 38
Simulated I/O Process ID 38
Signal to Send to Process 38
Reset Simulated I/O (9AH) 39

4 Examples

Overview 41
Introduction 41
Using Display Simulated I/O 42
To Open Display Simulated I/O: 42
To Write to Display Simulated I/O: 43
To Position the Cursor: 44
To Clear the Display: 45
To Close Display Simulated I/O: 46
Using Keyboard Simulated I/O 46
To Open Keyboard Simulated I/O: 46
Reading From the Keyboard: 48
To Close the Keyboard Interface: 49
Running The Simulated I/O Demo Program 49
Copying the Simulated I/O Demo Program 50

Compiling the Simulated I/O Demo Program 50
Copying the Default Emulator Configuration File 50
Entering the Softkey Interface 51
Configuring for Simulated I/O 51
Loading the Absolute File 52
Displaying Simulated I/O 52
Running the Demo Program 53
Modifying the Keyboard to Simulated I/O 53
Using Emulation Commands while Simulated I/O is Running 55
Closing the Keyboard Simulated I/O Demo 55
RS-232 Simulated I/O 69
Configuring RS-232 Lines for Simulated I/O 69
Serial Lines in UNIX Systems 69
Serial Lines Used with Simulated I/O 70
Serial Lines for User Terminals and Simulated I/O 70
Dedicated Serial Lines 71
Example Programs 72
Special Considerations 72

5 Error Codes

Overview 79
Introduction 79
Error Code Description (By Simulated I/O Command) 81
General Errors 81
Open (90H) Simulated I/O Errors 81
Open File Errors 82
Close (91H) Simulated I/O Errors 82
Close File Errors 82
Read (92H) Simulated I/O Errors 82
Read File Errors 82
Write (93H) Simulated I/O Errors 83
Read File Errors 83
Delete File (94H) Simulated I/O Errors 83
Position File (95H) Simulated I/O Errors 83
Position Cursor (96H) Simulated I/O Errors 84
Clear Display (97H) Simulated I/O Errors 84
UNIX Command (98H) Simulated I/O Errors 84
Kill Process (99H) Simulated I/O Errors 85

Introducing Simulated I/O

Overview

This chapter will answer the following questions:

- What is simulated I/O?
- Generally, how does simulated I/O work?
- What steps do I have to take to use simulated I/O?

Note

Simulated I/O is not available when your emulator has been restricted to real-time runs.

Simulated I/O Is ...

Simulated I/O is a process which allows your emulation system to communicate with host computer files, keyboard, and display.

Simulated I/O also allows your emulation system to execute UNIX commands; this means that, in addition to being able to communicate with your keyboard and display, your emulation system can also communicate with other host computer I/O devices, such as printers, plotters, modems, etc.

How Does Simulated I/O Work?

Communication between your emulation system and host computer files takes place through contiguous byte length emulation system memory locations. The first memory location is called the Control Address (CA). The Control Address and the memory locations which follow it are referred to as the CA buffer.

The Control Address (CA) buffer should be located in emulation RAM. It is possible, however, for the CA buffer to be located in target system RAM, but there will be a performance penalty for doing so. We recommend that you locate the Control Address and the CA buffer in emulation RAM.

Control Address buffers will never be more than 260 bytes in length because a maximum 256 bytes of information can be transferred between your emulation system and the host computer at a time. (Some simulated I/O commands require four additional bytes for command parameters.)

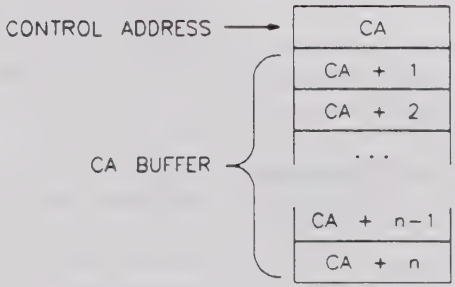
Communication between your emulation system and simulated I/O takes place as a series of requests by your emulation system and responses by simulated I/O.

User Program Request:

(1) Simulated I/O command parameters are first placed into the appropriate CA buffer locations.

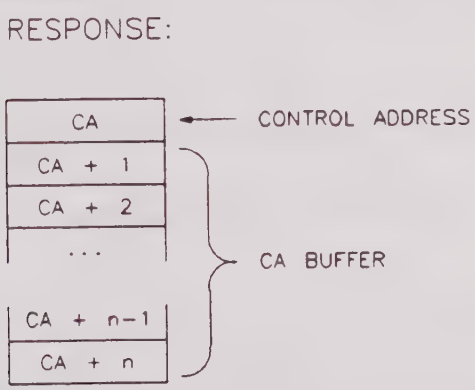
(2) After the CA buffer is loaded with simulated I/O command parameters, the command code is placed into the Control Address (CA) to cause the execution of the command.

USER PROGRAM REQUEST:



Response: (3) Before returning a value to the Control Address, some simulated I/O operations return additional information to the CA buffer.

(4) If the operation was successful, a 00H is returned to the Control Address. If the operation was not successful, an error code is returned to the CA.



Using Simulated I/O

The steps that you must take in order to use simulated I/O are listed below.

- You must write a program (to be executed by the emulator) which loads the CA buffer with the appropriate command parameters, places the appropriate command codes into the CA, and waits for the responses from simulated I/O. Command codes, command parameters, and simulated I/O responses are described in the "Simulated I/O Protocol" chapter.
- You must answer the simulated I/O emulation configuration questions. These questions are explained in the "Configuring for Simulated I/O" chapter.

The steps involved in using simulated I/O are also shown in the "Examples" chapter.

Configuring Simulated I/O

Overview

This chapter will:

- Explain the simulated I/O emulation configuration questions.
- List the restrictions on the use of simulated I/O.

Introduction

The simulated I/O subsystem must be set up by answering a series of configuration questions. These questions are a part of the general emulation configuration. They deal with enabling simulated I/O, setting the control address(es), and defining files used for standard I/O. The simulated I/O configuration questions are reached by answering *yes* to the following question.

Note

Simulated I/O is not available if the emulator is configured for real-time runs. When the emulator is configured for real-time runs, the following questions will not appear among the emulator configuration questions.

Answering The Simulated I/O Configuration Questions

Modify simulated I/O configuration? no (yes)

no — Answering no will cause the simulated I/O configuration questions to be skipped. The current simulated I/O configuration will not be modified.

yes — Answering yes causes the following questions, which will allow you to modify the simulated I/O configuration, to be asked.

Enable polling for simulated I/O?

no — Prevents the emulation software from reading the control address for simulated I/O commands. Answering no to this question will allow you to disable simulated I/O while maintaining the current simulated I/O configuration. Later, when you wish to enable simulated I/O, you may do so without having to re-enter control addresses or the file names for standard input, standard output, and standard error output. Answering no will also cause the rest of the simulated I/O questions to be skipped.

yes — Causes the emulation software to read the control address frequently to determine if the user program has requested any simulated I/O commands. Answering yes will also cause the following questions to be asked.

Simulated I/O control address 1? SIMIO_CA_ONE

Simulated I/O control address 2? SIMIO_CA_TWO

Simulated I/O control address 3? SIMIO_CA_THREE

Simulated I/O control address 4? SIMIO_CA_FOUR

Simulated I/O control address 5? SIMIO_CA_FIVE

Simulated I/O control address 6? SIMIO_CA_SIX

The symbol SIMIO_CA_ONE is the default symbol associated with the first simulated I/O Control Address. The default symbol may be replaced with any other valid symbol or an absolute address. If a symbol is specified, polling on that control address will not begin until a file containing that symbol is loaded. If an absolute address is specified, polling on that address will begin immediately.

The control address must be loaded into memory space assigned as RAM. It is recommended that the control address be located in emulation RAM since this allows user programs to run faster. Using target RAM causes the emulator to break into the monitor program every time the control address is polled for simulated I/O commands or data.

The following questions deal with the files associated with the three reserved file names "stdin", "stdout", and "stderr".

File used for standard input? /dev/simio/keyboard

File used for standard output? /dev/simio/display

File used for standard error output? /dev/simio/display

The default answers for these questions are "/dev/simio/keyboard", "/dev/simio/display", and "/dev/simio/display" respectively.

These files are not actually opened until Open (90H) is called with the file names "stdin", "stdout", and "stderr" and are only provided to allow easy redirection of input and output from the keyboard or display to a file, etc., without modifying the user program. (The compiler standard I/O libraries may open some or all of these reserved files automatically if simulated I/O is used. For more details, see the documentation on the simulated I/O libraries for the compiler in question.)

Enable simio status messages?

yes	The simulated I O command and return code are displayed in the upper right hand corner of the display.
no	Disables the simulated I/O status messages. Simulated I/O runs faster when the status messages are disabled.

Note

With HP 64700 firmware version 4.0 or greater, much of simulated I/O has been moved to the emulation card cage, and some simulated I/O operations complete without host intervention or notification (for example, reading from a file descriptor that has not been opened). With simulated I/O status messages enabled, there can be some simulated I/O commands that are not reported.

In addition, the simulated I/O write operation is now completed in the card cage, and any errors from write operations will not be reported on the write command that caused the error. Eventually, the host will report to the card cage that a write has failed and any write operations after this will receive the error.

Restrictions On Simulated I/O

The only two restrictions on the use of simulated I/O are:

- There is a limit of 12 open files at any one time.
- There may only be 4 active simulated I/O processes at any one time.

Since any simulated I/O file that is opened is associated with a file descriptor, opened files are independent of the Control Address. Up to 12 files may be opened with a single Control Address (CA). A total of 6 Control Addresses are allowed so that you can execute simulated I/O commands concurrently. Remember, a maximum of 12 simulated I/O files (between the 6 Control Addresses) may be opened at any one time.

Simulated I/O Protocol

Overview

This chapter will:

- Present the simulated I/O commands.
- Describe the simulated I/O command parameters.
- Describe the responses to simulated I/O commands.

Introduction

Communication between your emulation system and simulated I/O takes place as a series of requests by your emulation system to execute simulated I/O commands and responses by simulated I/O which tell your emulation system whether the command was successful or not. The communication between your emulation system and simulated I/O is done through byte length emulation system RAM locations (preferably emulation RAM, which is faster, but possibly target system RAM with less performance). All simulated I/O command codes and parameters are placed into memory locations relative to a Control Address (CA) memory location. (Control Addresses are assigned when configuring your emulation system for simulated I/O.) There are 10 simulated I/O commands:

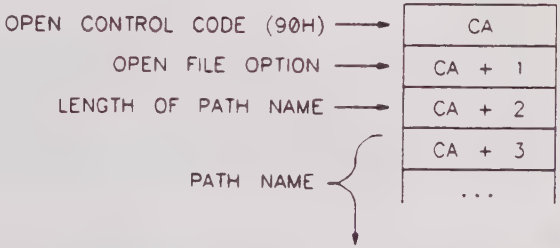
Command Code	Simulated I/O Command
90H	Open file
91H	Close file
92H	Read from file
93H	Write to file
94H	Delete a file
95H	Position file pointer
96H	Position cursor (display simulated I/O only).
97H	Clear display (display simulated I/O only).
98H	UNIX command
99H	Kill simulated I/O process.
9AH	Reset simulated I/O.

This chapter contains descriptions of the simulated I/O protocol, that is, the rules which govern the exchange of messages between your emulation system and simulated I/O.

Open File (90H)

Open (90H) opens a file or device for reading and/or writing by simulated I/O.

REQUEST TO OPEN FILE/DEVICE:



RESPONSE:

To invoke open, an open file option must be stored in address CA + 1, the length of the path name must be stored in location CA + 2, and the path name must be stored in CA + 3 and the following locations. Then, the open control code (90H) is stored in CA.

Note

The maximum number of simulated I/O files that may be open at any one time is 12.

Open File Option

The open file option stored in location CA + 1 must be one or more of the following flags OR-ed together:

Note

Exactly one of the first three open option flags must be used.

Read Only (code = 00H)

Write Only (code = 01H)

Read and Write (code = 02H)

Create (code = 04H)	The file is opened if it exists; otherwise, a new file is created and opened.
Append (code = 08H)	The current position is changed to the end of file prior to each write.
Truncate (code = 10H)	If the file exists, its length is truncated to 0.
Exclusive (code = 20H)	If both this flag and the create flag are set, open will fail if the file exists. It has no effect if the create flag is not set.
No Delay (code = 40H)	Read requests on this file will return immediately. With ordinary files this has no effect, but with devices such as a tty, emulation will not wait until data is available.' This flag may affect subsequent reads and writes. See open (2), read (2), and write (2) in the UNIX Reference manual for more details.

Open will fail if the file does not exist unless the create option is set. The open will succeed if the create option is set even if the file already exists, unless the exclusive option is also set. Upon opening, the file position pointer is set to the beginning of the file unless the append option is used where it points to the end of the file.

Length of Path Name

The length of the path name placed into location CA + 2 is the number of bytes in the path name where each (ASCII) character represents one byte. The "path name" can be shorter than the "length of path name" if the path name is null terminated (null = 00H). The path name need not be null terminated if its length is exact. (A null will be added after the specified number of characters.)

Path Name

The files "stdin", "stdout", and "stderr" are reserved for simulated I/O. When files with these names are opened, the file specified at configuration time for the standard I/O name requested will be the file actually opened. NOTE: unlike commands executed under a UNIX shell, these files are NOT automatically opened, but must be opened by your program.

The file names "/dev/simio/keyboard" and "/dev/simio/display" are also reserved. They represent devices for simulated I/O keyboards and displays. These file names are interpreted internally by the emulation software and do not exist as actual UNIX files. See examples of display and keyboard simulated I/O in the "Examples" chapter.

Absolute path names must be specified for the file names reserved for simulated I/O. Relative path names, as well as absolute path names, may be specified for other simulated I/O files.

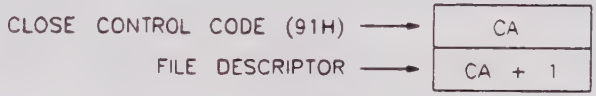
File Descriptor

The file descriptor returned by **open** should be saved for use by all reads, writes, seeks, or closes on the opened file. All simulated I/O commands which require a file descriptor specify that the file descriptor be in location CA + 1, and none of these commands modify the location CA + 1. Therefore, the file descriptor returned by open need not be saved if a unique Control Address is used for each open file or device, and the value at location CA + 1 is never modified from the time the file is opened until the file is closed. The file descriptor is one created and managed by emulation software and is not an actual UNIX file descriptor.

Close File (91H)

Close (91H) closes the file or device associated with the file descriptor specified. The file descriptor of the desired file is stored in CA + 1, and 91H is then stored in CA to invoke the close command. Zero is returned in the CA if the operation was successful, otherwise an error code is returned indicating why it failed.

REQUEST TO CLOSE FILE/DEVICE:



RESPONSE:

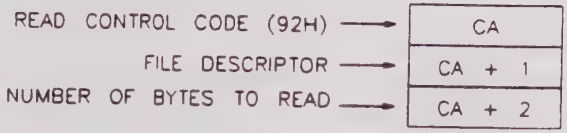
File Descriptor

The file descriptor placed into CA + 1 is the file descriptor that was returned by simulated I/O when the file was originally opened.

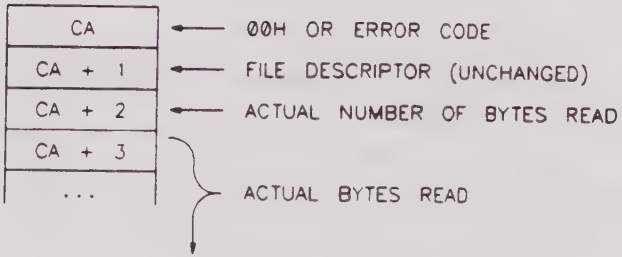
Read from File (92H)

To initiate a read operation, first store the file descriptor of the open file into CA + 1. Then, store the number of bytes (maximum of 255) to read in location CA + 2. A buffer large enough for the bytes requested must be provided starting at location CA + 3. Finally, store the read control code (92H) into location CA to initiate the read.

REQUEST TO READ FILE/DEVICE:



RESPONSE:



When the read operation is complete, the number of bytes actually read is returned to location CA + 2, the bytes read are stored in the buffer starting at location CA + 3, and 00H is returned to location CA if the operation was successful, or a nonzero error code is returned to location CA indicating why the operation failed.

Read behaves differently for the simulated keyboard device. The read will return only after a line has been completed (a <RETURN> or one of the softkeys has been pressed). The read will never return more than a single line of input. (A single line will return unless the number of characters requested is smaller than the length of the input; multiple reads will be required in this case, but a single read will never return text from more than one line, even if more than one line of text is available.) This will

generally be less than the requested number of characters. See the keyboard simulated I/O example in the "Examples" chapter.

File Descriptor

The file descriptor placed into CA + 1 is the file descriptor that was returned by simulated I/O when the file was originally opened.

Number of Bytes to Read

The number of bytes to read is placed in location CA + 2. The maximum number of bytes that can be read per read command is 255. Make sure that the CA buffer is big enough to hold the number of bytes that you wish to read.

Actual Number of Bytes Read

The end of file is indicated when 00H is returned to CA + 2. No error is indicated if fewer than the specified number of bytes are actually read. (This can happen when there are fewer than the specified number of bytes remaining in the file and, in most cases, when reading from the keyboard.)

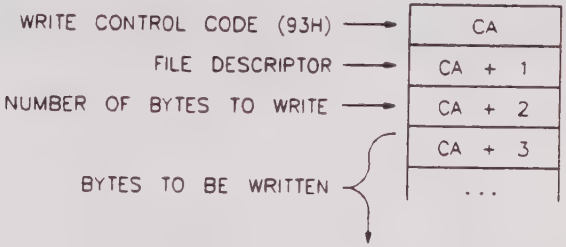
Actual Bytes Read

The bytes that were read from the file/device are placed into CA buffer locations starting at location CA + 3. A maximum number of 255 bytes can be read.

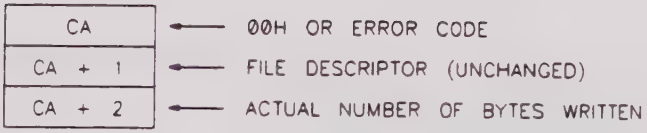
Write to File (93H)

To initiate a **write** operation, a buffer containing the bytes to be written must be set up at location CA + 3. The actual number of bytes to write (maximum of 255) must be stored in location CA + 2, and the file descriptor of an open file must be stored in location CA + 1. Finally, the **write** control code (93H) must be stored in location CA.

REQUEST TO WRITE TO FILE/DEVICE:



RESPONSE:



When the **write** operation is complete, 00H is returned to location CA if the write was successful, and the actual number of bytes written is returned to location CA + 2 (the file descriptor in CA + 1 is unchanged); otherwise, an error code is returned to location CA indicating why the operation failed, and the number of bytes written is set to 00H.

If the file was opened with the "append" option, the file pointer is positioned to the end of file prior to each write. (NOTE: the "append" option is ignored if the file is the display.)

File Descriptor

The file descriptor placed into CA + 1 is the file descriptor that was returned by simulated I/O when the file was originally opened.

Number of Bytes to Write

When requesting to write, the number of bytes to write is placed into location CA + 2. The maximum number of bytes that can be written per write command is 255. Make sure that the CA buffer is big enough to hold the number of bytes you wish to write.

Bytes to be Written

The bytes to be written are placed into the CA buffer starting at location CA + 3. The maximum number of bytes that can be written is 255.

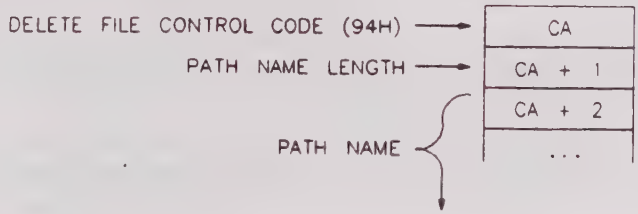
Actual Number of Bytes Written

If the write operation was successful, simulated I/O will tell you how many bytes were actually written by placing their number into location CA + 2.

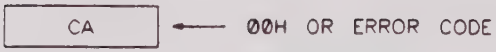
Delete a File (94H)

Delete File (94H) removes the directory entry named. This action is invoked by storing the length of the file name in location CA + 1, the file name (relative path or absolute path) starting at location CA + 2, and then storing the delete file control code (94H) into the CA.

REQUEST TO DELETE A UNIX FILE:



RESPONSE:



Path Name Length

The length of the path name placed into location CA + 1 is the number of bytes in the path name where each (ASCII) character represents one byte. The "path name" can be shorter than the "path name length" if the path name is null terminated (null = 00H). The path name need not be null terminated if its length is exact. (A null will be added after the specified number of characters.)

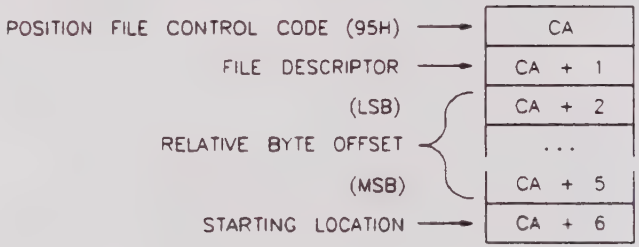
Path Name

Note that the file names for delete file are not interpreted by simulated I/O; therefore, special files such as "stdin" or "/dev/simio/display" are treated as standard UNIX files. For example, trying to delete a file called "stdin" will attempt to to remove the file "./stdin", not the special simulated I/O "stdin" file.

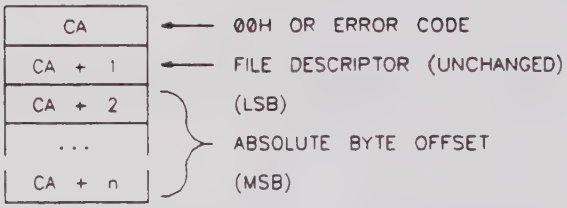
Position File Pointer (95H)

To initiate a **position file** operation, store the file descriptor into CA + 1, the starting code (see below) into location CA + 6, and a 32 bit SIGNED integer into locations CA + 2 through CA + 5. The 32 bit signed integer will indicate a byte offset from the starting code. The least significant byte of the offset will be in location CA + 2, and the most significant byte will be in location CA + 5. After the file descriptor, the byte offset, and the starting code are placed in the CA buffer, store the position file control code (95H) into CA.

REQUEST TO CHANGE FILE POSITION POINTER:



RESPONSE:



When the operation is complete, the absolute offset from the beginning of the file is returned into locations CA + 2 through CA + 5 (a 32 bit SIGNED integer whose least significant byte is in location CA + 2 and whose most significant byte is in location CA + 5), and 00H is returned to CA if the operation was successful or an error code is returned to CA if the operation failed.

Note

If the file was opened with the "append" option, the position pointer will automatically be repositioned to the end of file before any write operation, even if you use a position command before the write. You can, however, use the position file command followed by a read command to read from a specific position in a simulated I/O file that was opened with the "append" option.

This command has no effect on the special simulated I/O files "/dev/simio/keyboard" and "/dev/simio/display", and if it is used, an error code for INVALID COMMAND (09H) will be returned.

FILE DESCRIPTOR. The file descriptor placed into CA + 1 is the file descriptor that was returned by simulated I/O when the file was originally opened.

Relative Byte Offset

The relative byte offset is a 32 bit SIGNED integer which indicates a byte offset from the starting location. The relative byte offset is placed into locations CA + 2 through CA + 5 (with the least significant byte in location CA + 2 and the most significant byte in location CA + 5).

Starting Location

The starting location placed in location CA + 6 will determine the position to which the "relative byte offset" will be added to determine the resulting pointer position. The codes which may be placed in the "starting location" are:

starting code = 00H	The offset is from the beginning of the file.
starting code = 01H	The offset is from the current position.
starting code = 02H	The offset is from the end of the file.

To **rewind** a file, use starting code = 00H and offset = 00H. To **find the current position** in the file, use the starting code = 01H and offset = 00H (the current location is returned in the "absolute offset" from the beginning of the file.

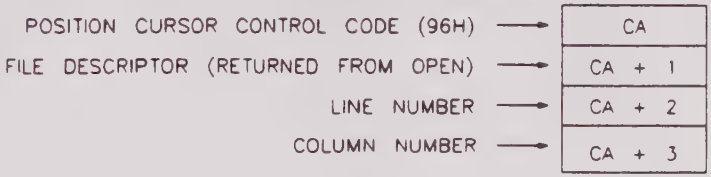
Absolute Byte Offset

The absolute byte offset returned to locations CA + 2 through CA + 5 is a 32 bit SIGNED integer whose least significant byte is in location CA + 2 and whose most significant byte is in location CA + 5.

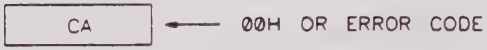
Position Cursor on Display (96H)

To position the cursor for writes to the screen, store the file descriptor returned from the open command in location CA + 1, the desired line number (0 through 49) in location CA + 2, and the desired column number (0 through 79 or one minus the number of columns on the display device if greater than 80) into location CA + 3. Then, store the position cursor control code (96H) into CA. The next write will begin at this location. Zero is returned to location CA if the position cursor operation is successful; otherwise, a nonzero error code is returned to CA.

REQUEST TO POSITION CURSOR:



RESPONSE:



File Descriptor (Returned from Open)

The file descriptor that is placed into location CA + 1 is the file descriptor that was returned when the file was originally opened.

Line Number

The line number placed into location CA + 2 must be one of the 50 lines of display simulated I/O (0 through 49).

Column Number

The column number placed into location CA + 3 must be 0 through 79 (or one minus the number of columns on the display device if greater than 80).

Clear Display (97H)

To clear the display, the file descriptor returned from open must be stored into location CA + 1, the clear display control code (97H) is stored into location CA. Zero is returned to location CA if the operation was successful; otherwise, a nonzero error code is returned to CA. When the display is cleared, the cursor is left at column 0, line 0.

REQUEST TO CLEAR THE DISPLAY:

RESPONSE:

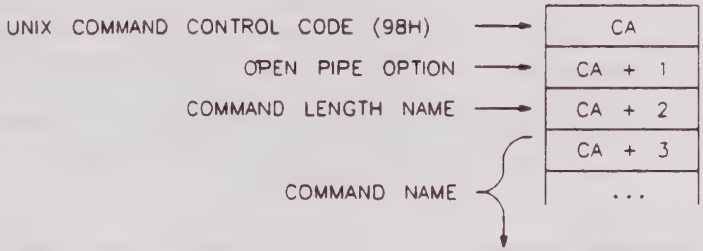
File Descriptor (Returned from Open)

The file descriptor that is placed into location CA + 1 is the file descriptor that was returned when the file was originally opened.

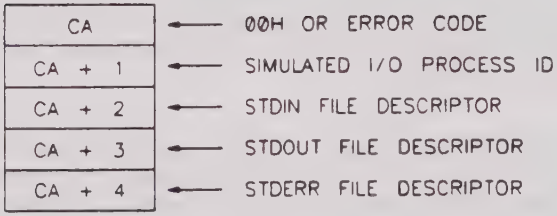
UNIX Command (98H)

UNIX Command (98H) allows you to execute a UNIX command from emulation. In order to execute a UNIX command, emulation forks off a process which then opens pipes to the first three file descriptors (the requested command's standard in, out, and error). This forked process will then execute the requested command in a sub shell with communication (if requested) to the shell's "stdio".

REQUEST TO EXECUTE UNIX COMMAND:



RESPONSE:



Note

A maximum number of 4 simulated I/O processes can be active at any one time.

If any pipes are opened to the simulated I/O process (i.e., bits in the Open Pipe Option are SET), they are considered to be open files. A maximum of 12 open files are allowed at any one time.

Open Pipe Specification

The pipes to open are specified as a bitmap in location CA + 1.

- Bit 0 set specified "stdin" to be opened.
- Bit 1 set specifies "stdout" to be opened.
- Bit 2 set specifies "stderr" to be opened.

If any of the three bits is cleared, " dev/null" will be opened. Inputs to the UNIX command can be sent by using the Write (93H) command and the "stdin" file descriptor which is returned to location CA + 2. Likewise, outputs from the UNIX command can be read by using the Read (92H) command and the "stdout" and "stderr" file descriptors which are returned to locations CA + 3 and CA + 4, respectively.

Command Name Length

The length of the command name placed into location CA + 2 is the number of bytes in the command name where each (ASCII) character represents one byte. The "command name" can be shorter than the "command name length" if the name is null terminated (null = 00H). The command name need not be null terminated if its length is exact. (A null will be added after the specified number of characters.)

Command Name

The command name is executed as an argument to "sh" executed with the "-c" option so any valid shell command line can be specified and shell features such as I/O redirection, pipes, and filename expansion using "*", "?", etc., can be used. Thus, the strings like "pr | lpr" and "ls *.c > source_files" are valid command names.

Simulated I/O Process ID

The "simulated I/O process ID" returned to location CA + 1 is a process ID internal to simulated I/O and, therefore, does not directly correspond to a UNIX process ID. This allows the ID to be contained in one byte.

Stdin File Descriptor

If the open pipe option specifies that "stdin" be opened, the descriptor of the file that is opened is returned to location CA + 2.

Any inputs to the UNIX command may be written to the "stdin" file by using the simulated I/O Write (93H) command, just as any other simulated I/O file is written to.

Stdout File Descriptor

If the open pipe option specifies that "stdout" be opened, the descriptor of the file that is opened is returned to location CA + 3. Any outputs from the UNIX command can be read from the "stdout" file by using the simulated I/O Read (92H) command, just as any other simulated I/O file is read from.

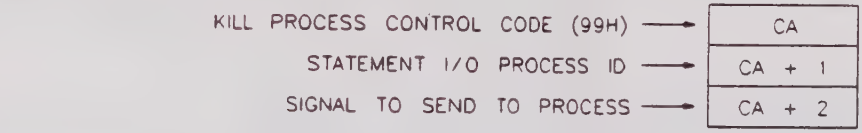
Stderr File Descriptor

If the open pipe option specifies that "stderr" be opened, the descriptor of the file that is opened is returned to location CA + 4. Any error outputs from the UNIX command can be read from the "stderr" file by using the simulated I/O Read (92H) command, just as any other simulated I/O file is read from.

Kill Simulated I/O Process (99H)

Kill (99H) can be called to terminate the execution of a UNIX process started under simulated I O. This command sends the signal specified in CA + 2 to the requested process.

REQUEST TO KILL SIMULATED I/O PROCESS:



RESPONSE:

This command can also be used to determine if a process is still running by sending a signal 00H to the process. A return value of 00H indicates that the process is still active and a nonzero return value indicates that the process no longer exists.

Simulated I/O Process ID

The simulated I/O process ID placed into location CA + 1 is the process ID that was returned from the simulated I/O UNIX Command (98H) operation.

Signal to Send to Process

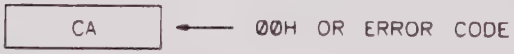
See **signal (2)** in the UNIX reference manual for information on the signal to send to the process.

Reset Simulated I/O (9AH)

Reset simulated I/O (9AH) closes all open simulated I/O files and removes all entries from the simulated I/O process table. The processes are not explicitly killed, but SIGPIPE is sent to each process connected by a pipe to simulated I/O when the pipe is closed; this, by default, kills the process. This command is provided to allow restarting a program running on an emulated processor without having side effects from previously opened files or previously executed processes.

REQUEST TO RESET SIMULATED I/O PROCESS:

RESPONSE:



Examples

Overview

This chapter will:

- Show you how to use display simulated I/O.
- Show you how to use keyboard simulated I/O.
- Step you through the process of running the simulated I/O demo program.

Introduction

Simulated I/O lets you use your workstation display and keyboard as emulation system output and input devices. This chapter will show you how to use the simulated I/O commands for display and keyboard simulated I/O.

This chapter also contains a "C" demo program which uses display and keyboard simulated I/O, and other simulated I/O commands as well. The demo program in this chapter has been included with the Softkey Interface emulation software. The examples at the end of this chapter step you through the process of compiling, linking, and running the demo program in emulation.

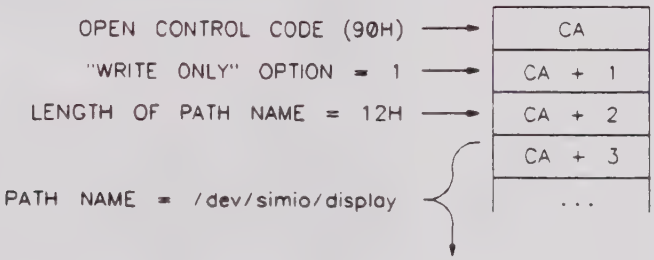
Using Display Simulated I/O

The simulated I/O display interface allows your emulation system program to access the workstation display. The use of the simulated display does not actually put anything on the display until activated by pressing the "display" then "sim_io" softkeys. Instead, the characters are stored in a buffer and displayed when the simulated I/O display is activated. This allows you to use the emulation "display" command to display memory, registers, etc., while your display simulated I/O program is running. The display buffer can always be updated (by a write or clear) whether it is currently being displayed or not. No data is lost if the workstation display is being used for other emulation commands while your simulated program is writing to display simulated I/O.

To Open Display Simulated I/O:

To open the display, the standard simulated I/O open command is used. The file descriptor returned by "open" must be saved for all accesses to the display. The special file name "/dev/simio/display" is reserved for use by the simulated I/O display. This file does not correspond to any actual UNIX file, but instead is internal to the emulation software.

TO OPEN DISPLAY SIMULATED I/O:



RESPONSE:

The display opened is a 50 line by at least 80 column buffer. (The actual number of columns is 80 or the number of columns on the display device, whichever is greater.) The display buffer has associated with it a current cursor location, and all writes begin at this current cursor location. When the display is open, the cursor is located at the upper left corner of the display buffer (row 0, column 0), and the cursor position is updated after each write and position cursor command.

Only 16 lines of the buffer can be displayed at one time on the standard 24 line display; the actual number of lines displayed will be the number of lines on the display device minus 2 lines for the simulated I/O header and 6 lines for the status, command and softkey lines.

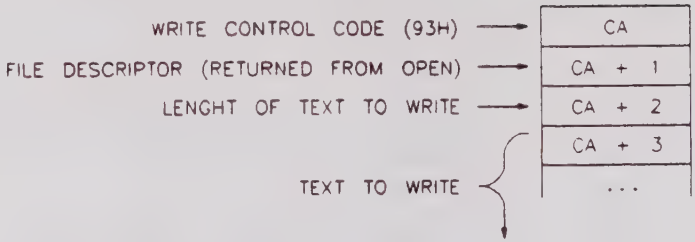
The <NEXT PAGE>, <PREV PAGE>, <ROLL UP>, and <ROLL DOWN> keys can be used to position the display anywhere in the buffer when the simulated I/O display is the active emulation display. (But, note that the display is updated to include the current cursor position in the visible region after each **write** or **position cursor** command.)

Any write past the last column will automatically wrap to the next line, and any write past the last column on the 50th line will cause all the lines to be scrolled up by one.

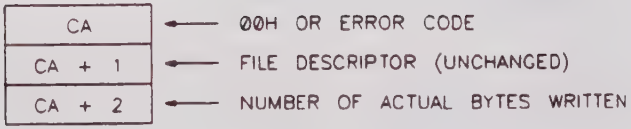
To Write to Display Simulated I/O:

To write to the display, the standard simulated I/O write command is used. The file descriptor returned from the open command must be stored in location CA + 1. A buffer containing the text to be written must be set up starting at location CA + 3. The length of the text must be stored in location CA + 2; then, the write control code (93H) must be written to location CA. Zero is returned to the control address, and the number of bytes actually written is returned to location CA + 2 if the write was successful; a nonzero error code is returned to location CA if the write operation was not successful.

TO WRITE TO DISPLAY SIMULATED I/O:



RESPONSE:



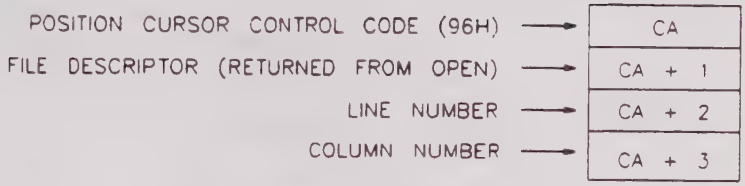
If the text to be written spans more than one line (contains more characters than the number of columns or contains a newline character), the writing will continue on the next line at column 0.

Text is written to the current cursor position, and the cursor position is updated after each write.

To Position the Cursor:

To position the cursor for writes to the screen, store the file descriptor returned from the open command in location CA + 1, the desired line number (0 through 49) in location CA + 2, and the desired column number (0 through 79 or one minus the number of columns on the display device if greater than 80) in location CA + 3. Then, store the position cursor command code (96H) in location CA. The next write will begin at this location. Zero is returned to location CA if the position command is successful; a nonzero error code is returned otherwise.

TO POSITION THE CURSOR:



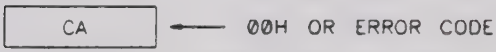
RESPONSE:

To Clear the Display: To clear the display, the file descriptor returned from open must be stored in location CA + 1, the clear display command code (97H) is stored in location CA. Zero is returned to location CA if the command was successful; otherwise, a nonzero error code is returned to location CA.

The display is cleared, and the cursor is left at row 0, column 0.

TO CLEAR THE DISPLAY:

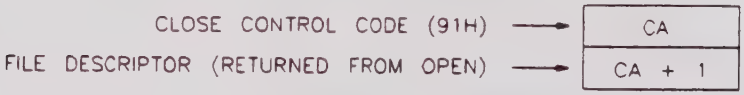
RESPONSE:



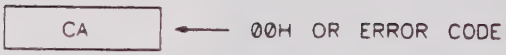
To Close Display Simulated I/O:

To close the display, the standard simulated I/O close command is used. The file descriptor returned from the open command must be placed in location CA + 1; then, the close command code (91H) is placed into location CA. Zero is returned to CA if the close is successful; otherwise, a nonzero error code is returned.

TO CLOSE DISPLAY SIMULATED I/O:



RESPONSE:



Using Keyboard Simulated I/O

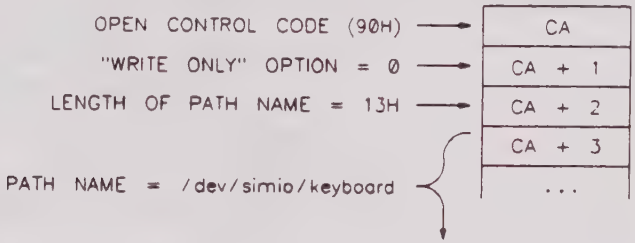
The simulated I/O keyboard interface allows your emulation system program to access the workstation keyboard as an input device.

To Open Keyboard Simulated I/O:

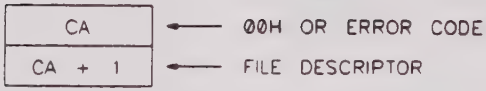
To open the simulated I/O keyboard, use the standard simulated I/O open command. The open option code stored in location CA + 1 should be 0 for read only. Simulated I/O uses the special file name "/dev/simio/keyboard" to indicate the keyboard interface; this name must be stored in the CA buffer starting at location CA + 3, and the length of the name (13H) must be stored in location CA + 2. The open file command code must be stored in location CA. Zero is returned to location CA if the open is successful; otherwise a nonzero error code is returned. A file descriptor is

returned by the open command to location CA + 1 if the open is successful.

TO OPEN KEYBOARD SIMULATED I/O:



RESPONSE:



This file descriptor must be saved for use by all other commands accessing the keyboard.

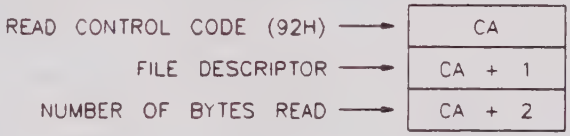
Opening the keyboard does not actually direct keyboard characters to simulated I/O until you activate the simulated I/O keyboard by pressing the "modify" softkey followed by the "keyboard_to_simio" softkey. Modifying the keyboard to simulated I/O clears the command line area of the workstation display and begins storing the keyed in lines for reading by the simulated I/O system.

All characters input from the keyboard are displayed in the command line area of the display as they are being typed. The command line can be edited using the standard command line editing facilities before <RETURN> is pressed. The input line can be terminated by pressing <RETURN>, any of the softkeys, or <CTRL>**d**. No characters can be read by the simulated I/O program until <RETURN>, one of the softkeys, or <CTRL>**d** is pressed.

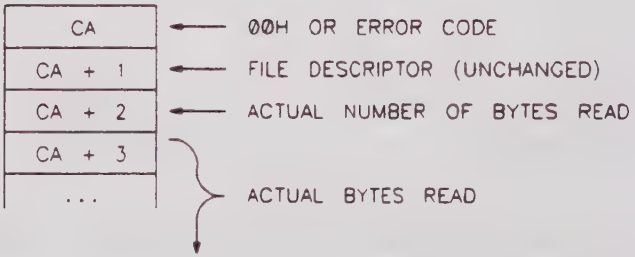
Reading From the Keyboard:

To read from the keyboard, the standard simulated I/O read command is used.

READING FROM THE KEYBOARD:



RESPONSE:

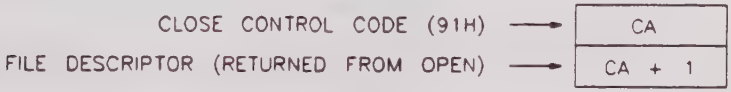


The keyboard input is saved as a list of lines, and each call to read will read a maximum of one line of text. (The actual number of characters read will be returned.) The last characters of the line read will be a newline, which indicates that the input line was terminated by a <RETURN> or an escape character and an ASCII 1 through 8, indicating one of the softkeys f1 through f8 was used to terminate the input line. (Note that f1 corresponds to the "suspend" softkey.) If the keyboard input line was terminated with a <CTRL>**d**, no characters are added to the line. If the number of characters to read is less than the length of the line, multiple calls to read will be required to read the complete line, and no characters will be lost. The maximum length of a keyboard input line is 240 characters; therefore, a complete line will always be read if the number of characters requested in the read command is 240.

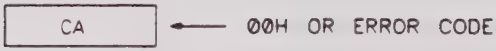
To Close the Keyboard Interface:

To close the keyboard interface, the standard simulated I/O close command is used. The file descriptor returned from the open command must be placed in location CA + 1; then, the close command code (91H) is placed into location CA. Zero is returned to CA if the close is successful; otherwise, a nonzero error code is returned.

TO CLOSE THE KEYBOARD:



RESPONSE:



Running The Simulated I/O Demo Program

The simulated I/O demo program which follows is written in the "C" language. This section will step you through the process of compiling and linking the demo program and running it in emulation.

The examples that follow have been created in the HP 64742 68000 emulation environment. The choice to run the demo program in the HP 64742 emulation environment is purely arbitrary, and most of the instructions that follow will be very similar, if not the same, for whichever emulation system you choose run this demo in.

Copying the Simulated I/O Demo Program

The simulated I/O demo program shown in figure 4-3 has been included with your emulation software and is located in the directory /usr/hp64000/demo/emul/simio. To copy the demo program to your directory, enter the commands shown below.

(The period just before the <RETURN> specifies that the file will have the same name in your current directory.)

```
$ cp
/usr/hp64000/demo/emul/simio/simiodemo.h .
<RETURN>
$ cp
/usr/hp64000/demo/emul/simio/simiodemo.c .
<RETURN>
```

Compiling the Simulated I/O Demo Program

To compile the simiodemo.c demo program with the AxLS (Advanced Cross Language System) 68000 C Cross Compiler, enter the following command.

```
$ cc68000 -vOGNr hp64742 -o simiodemo
simiodemo.c <RETURN>
```

(There is one duplicate symbol error when compiling the program this way, but the compiler ignores the symbol that should be ignored, and the generated absolute file works correctly.)

Notice that the compiler's "-N" (no I/O) option is used. This is because the demo program contains routines that use simulated I/O. When the "-N" option is not used, the compiler links in its own I/O routines which use simulated I/O.

Copying the Default Emulator Configuration File

Since the AxLS 68000 C Cross Compiler provides default configuration files for the HP 64742 68000 Emulator, copy the default emulator configuration file to the current directory before you enter the emulation system.

```
$ cp /usr/hp64000/env/hp64742/config.EA
config.EA <RETURN>
```

Entering the Softkey Interface

If you have installed your emulator and Softkey Interface software, opened a window, and set and exported the proper environment variables as directed in the *Softkey Interface Installation Notice*, you can enter the Softkey Interface with the following command:

```
$ emul700 <emul_name> <RETURN>
```

The "emul_name" in the command above is the logical emulator name given in the HP 64700 emulator device table (/usr/hp64000/etc/64700tab.net).

Configuring for Simulated I/O

First, load the compiler's emulator configuration file.

```
load configuration config <RETURN>
```

Now, modify the configuration to enable the simulated I/O feature and to give the Softkey Interface the demo program's control address.

```
modify configuration <RETURN>
```

Now step through the emulation configuration questions, by pressing the <RETURN> key, until you come to the question:

```
Modify simulated I/O configuration? yes
<RETURN>
```

Answer the simulated I/O configuration questions as shown below.

```
Enable polling for simulated I/O? yes
<RETURN>
Simulated I/O control address 1?
SIMIO_CA_ONE control_addr <RETURN>
Simulated I/O control address 2?
SIMIO_CA_TWO <RETURN>
Simulated I/O control address 3?
SIMIO_CA_THREE <RETURN>
Simulated I/O control address 4?
SIMIO_CA_FOUR <RETURN>
Simulated I/O control address 5?
SIMIO_CA_FIVE <RETURN>
Simulated I/O control address 6?
SIMIO_CA_SIX <RETURN>
```

```
File used for standard input?
/dev/simio/keyboard <RETURN>
File used for standard output?
/dev/simio/display <RETURN>
File used for standard error output?
/dev/simio/display <RETURN>
Enable simio status messages? yes
```

Press <RETURN> for the rest of the emulation configuration questions until you are asked the name of the configuration command file. Answer as shown below.

```
Command file name? simiodemocfg <RETURN>
```

Simulated I/O is now configured for the demo program. Notice that the demo program only uses one CA. The name "control_addr" is the symbol associated with the "control_addr" array defined in the **simiodemo.c** program.

Loading the Absolute File

To load the absolute file, enter the following command:

```
load simiodemo.x <RETURN>
```

Displaying Simulated I/O

To display the contents of the display simulated I/O buffer, enter the command shown below.

```
display simulated_io <RETURN>
```

The simulated I/O display is shown in figure 4-1.

```
 Simulated I/O display




STATUS:  M68000--Running in monitor_____________________________...R....
_display  simulated_io

  run     trace     step   display          modify   break     end    ---ETC--
```

Figure 4-1. Displaying Simulated I/O

Running the Demo Program

To run the simulated I/O demo program, enter the following command:

```
run from entry <RETURN>
```

The symbol "entry" was created by the "C" library which was linked with the demo program. The message "KEYBOARD NOW OPEN" will be written to the display simulated I/O buffer and will be displayed on the screen.

Modifying the Keyboard to Simulated I/O

To be able to use your workstation keyboard as the standard input to the simulated I/O demo program, you must first activate your keyboard with the command shown below.

```
modify keyboard_to_simio <RETURN>
```

Your keyboard is now active as the simulated I/O input device as you can tell by the appearance of the suspend softkey. You can now

enter text on the command line using the standard command line editing features.

```
Type in a line of text on the command line.
<RETURN>
```

The text that you just entered will appear on the simulated I/O display (see figure 4-2). Reads of the simulated I/O keyboard will only take place after the <RETURN> key or one of the softkeys has been pressed.

```
Simulated I/O display                  Simulated I/O command: read
display is open                              Return code: 00H
KEYBOARD NOW OPEN
OUTPUT: Type in a line of text on the command line.




STATUS:  M68000--Running user program_____________________________.........
-

suspend  _______ _______ _______        _______ _______ _______ _______
```

Figure 4-2. Entering Text from the Keyboard

Pressing the "suspend" softkey will not close the simulated I/O keyboard; it will, however, deactivate your workstation keyboard as the simulated I/O input device, and allow you enter emulation commands.

Using Emulation Commands while Simulated I/O is Running

It is possible to enter emulation commands while your simulated I/O program is running. For example, enter the commands shown below.

```
suspend
trace <RETURN>
display trace <RETURN>
display simulated_io <RETURN>
modify keyboard_to_simio <RETURN>
```

You once again displaying simulated I/O and your workstation keyboard is again the active simulated I/O input device. You can enter text followed by the <RETURN>, and the demo program will move the text to the simulated I/O display buffer.

Closing the Keyboard Simulated I/O Demo

The simulated I/O demo program will deactivate the keyboard as an input device when the following key is pressed:

```
<RETURN>
```

Keyboard simulated I/O is now closed, and the rest of the simulated I/O demo program will begin executing. You will see the simulated I/O demo program test the "position cursor" command by selectively writing to a single display location. Then, the execute "UNIX command" is used to concatenate the **simiodemo.c** demo program to the simulated I/O "stdout" file (which was specified as "/dev/simio/display" in the simulated I/O configuration). Other simulated I/O commands are executed as well. Look at the **simiodemo.c** file (figure 4-3) to see exactly what the simulated I/O demo program does.

```
#include "simiodemo.h"

#define READ_BUF_SIZE 255
#define TRUE 1
#define FALSE 0

/* FORWARD DECLARATIONS */
extern int initsimio();
extern int open();
extern int close();
extern int read();
extern int write();
extern int clear_screen();
extern int pos_cursor();
extern int exec_cmd();
extern int kill();
extern int unlink();
extern long lseek();

unsigned char control_addr[300];
unsigned char *simio_addr;
int errno;

main()
{
        int fd1, fd2, fd3;          /* Declarations for the file descriptors. */
        int fd4;
        int pid1, pid2;             /* Declarations for process IDs.          */
        int numb_read;
        unsigned char buf[READ_BUF_SIZE];

        simio_addr = control_addr;
        initsimio();

        /*
         *      Open stdout and stdin -- typically a display and
         *      keyboard.  These can be set up during configuration
         *      to be the defaults of /dev/simio/display and
         *      /dev/simio/keyboard or changed to a file or
         *      directly to another terminal (/dev/tty00 for example).
         */

        fd1 = open("stdout", S_O_WRITE);
        fd4 = open("stdin", S_O_READ | S_O_NDELAY);

        write(fd1, "KEYBOARD NOW OPEN\n", 18);

        /*
         *      This loop reads the keyboard until a line containing
         *      only a <RETURN> is encountered.
         *
         *      Each line read is written to the display following
         *      the string "OUTPUT:  ".
         */

        while (TRUE)
        {
```

Figure 4-3. The "simiodemo.c" Demo Program

```
        numb_read = read(fd4, buf, READ_BUF_SIZE);
        if (numb_read > 0)
        {
                /* keyboard input detected */
                write(fd1, "OUTPUT:  ", 9);
                write(fd1, buf, numb_read);
                if (*buf == '\n')
                {
                        break;
                }
        }
}

close(fd4);
write(fd1, "KEYBOARD NOW CLOSED\n", 20);
close(fd1);

/*   Examples of position cursor and clear screen commands.          */

fd1 = open("/dev/simio/display", S_O_WRITE);
pos_cursor(fd1, 5, 30);
write(fd1, "TESTING #1 POSITION CURSOR COMMAND", 34);
pos_cursor(fd1, 5, 39);
write(fd1, "2", 1);
pos_cursor(fd1, 5, 39);
write(fd1, "3", 1);
pos_cursor(fd1, 5, 39);
write(fd1, "4", 1);
clear_screen(fd1);
close(fd1);
fd2 = open("stdout", S_O_WRITE | S_O_CREATE);

/*
 *      Example of the CREATE and EXCL flags.
 *
 *      If the file "simiodemo.out" does not already exist
 *      the file is created and the message "File did not exist"
 *      is written to the file "simiodemo.out".  If the file
 *      already existed it is removed then created and the
 *      text "File already existed" is written to the file.
 *      If the file cannot be opened, the message "Cannot open
 *      simiodemo.out" is written to standard out.
 */

fd3 = open("simiodemo.out", S_O_RDWR | S_O_CREATE | S_O_EXCL);
if (fd3 < 0)
{
        if (errno == FILE_EXISTS)
        {
                unlink("simiodemo.out");
                fd3 = open("simiodemo.out", S_O_RDWR | S_O_CREATE | S_O_EXCL);
                write(fd3, "File already existed\n", 21);

        }
        else
        {
                write(fd2, "Cannot open simiodemo.out", 25);
        }
```

Figure 4-3. The "simiodemo.c" Demo Program (Cont'd)

```
}
else
{
        write(fd3, "File did not exist\n", 19);
}

/*
 *      Example of the position file command. -- This assumes that the
 *      file "simiodemo.out" was successfully opened and written to
 *      by the code above.
 */

/*      Position from beginning of file.                                */

lseek(fd3, 0L, 0);
if ((numb_read = read(fd3, buf, READ_BUF_SIZE)) > 0)
{
        write(fd2, buf, numb_read);
}

lseek(fd3, 1L, 0);
if ((numb_read = read(fd3, buf, READ_BUF_SIZE)) > 0)
{
        write(fd2, buf, numb_read);
}

lseek(fd3, 2L, 0);
if ((numb_read = read(fd3, buf, READ_BUF_SIZE)) > 0)
{
        write(fd2, buf, numb_read);
}

/*      Offset from the current position.
 *      -- Note that this is the end of file
 *      because of the reads which read to end of file.
 */

lseek(fd3, -1L, 1);
if ((numb_read = read(fd3, buf, READ_BUF_SIZE)) > 0)
{
        write(fd2, buf, numb_read);
}

lseek(fd3, -2L, 1);
if ((numb_read = read(fd3, buf, READ_BUF_SIZE)) > 0)
{
        write(fd2, buf, numb_read);
}

lseek(fd3, -3L, 1);
if ((numb_read = read(fd3, buf, READ_BUF_SIZE)) > 0)
{
        write(fd2, buf, numb_read);
}

/*      Position from end of file.                                      */

lseek(fd3, -1L, 2);
```

Figure 4-3. The "simiodemo.c" Demo Program (Cont'd)

```
if ((numb_read = read(fd3, buf, READ_BUF_SIZE)) > 0)
{
        write(fd2, buf, numb_read);
}

lseek(fd3, -2L, 2);
if ((numb_read = read(fd3, buf, READ_BUF_SIZE)) > 0)
{
        write(fd2, buf, numb_read);
}

lseek(fd3, -3L, 2);
if ((numb_read = read(fd3, buf, READ_BUF_SIZE)) > 0)
{
        write(fd2, buf, numb_read);
}

/*
 *      Example of using the execute HP-UX command.
 *
 *      A pipe is connected to the standard output of the cat
 *      command.  This output is then written to the display.
 */

pid1 = exec_cmd("cat simiodemo.c", (int *) 0, &fd1, (int *) 0);

/*
 *      The sleep command is executed as an example of the
 *      kill command.  A kill with signal 0 (NULL signal)
 *      is issued to find the status of the sleep.  The NULL
 *      signal will not terminate the process but the return
 *      value of the kill routine indicated if the process
 *      exits or not.
 */

pid2 = exec_cmd("sleep 5", (int *) 0, (int *) 0, (int *) 0);

while ((numb_read = read(fd1, buf, READ_BUF_SIZE)) > 0)
{
        write(fd2, buf, numb_read);

        /* See if sleep is still alive.                                  */

        if (kill(pid2, 0) == 0)
        {
                write(fd2, "\nProcess still alive\n", 21);
        }
        else
        {
                write(fd2, "\nProcess terminated\n", 20);
        }
}

write(fd2, "\nDEMO COMPLETE", 14);

close(fd1);
close(fd2);
```

Figure 4-3. The "simiodemo.c" Demo Program (Cont'd)

```
        close(fd3);
}
int
initsimio()

/*
 *      This command is not actually required, but it allows
 *      the program to be stopped while simio files are still
 *      open and restarted without side effects from the previously
 *      opened files.
 *
 *      RETURNS          0 for no error.
 *                      -1 for errors during the simio reset command
 *                       and the global errno is set to the error code.
 */

{
        *simio_addr = S_RESET;

        while (*simio_addr == S_RESET)
        {
                /* Empty loop - wait for results */
        }

        if (*simio_addr == 0)
        {
                return 0;
        }
        else
        {
                errno = (unsigned int) *simio_addr;
                return -1;
        }
}
int
open(path, open_option)
char *path;
int open_option;

/*
 *      RETURNS          file descriptor >= 0 -- if the open succeeded.
 *                       -1 if the open failed, errno is set the the
 *                       error code.
 */

{
        unsigned char *addr_ptr;

        addr_ptr = simio_addr + 1;

        *addr_ptr = open_option;
        addr_ptr++;

        *addr_ptr = 255;
        addr_ptr++;

        while (*path != '\0')
```

Figure 4-3. The "simiodemo.c" Demo Program (Cont'd)

```
        {
                *addr_ptr = *path;
                addr_ptr++;
                path++;
        }

        *addr_ptr = '\0';

        *simio_addr = S_OPEN;

        while (*simio_addr == S_OPEN)
        {
                /* Empty loop - wait for results */
        }

        if (*simio_addr == 0)
        {
                return (unsigned int) *(simio_addr + 1);
        }
        else
        {
                errno = (unsigned int) *simio_addr;
                return -1;
        }
}
int
close(file_des)
int file_des;

/*
 *      RETURNS          0 for no error.
 *                      -1 for error and the global errno is set to the error code.
 */

{
        *(simio_addr + 1) = file_des;
        *simio_addr = S_CLOSE;

        while (*simio_addr == S_CLOSE)
        {
                /* Empty loop - wait for results */
        }

        if (*simio_addr == 0)
        {
                return 0;
        }
        else
        {
                errno = (unsigned int) *simio_addr;
                return -1;
        }
}
int
read(fd, buffer, nbytes)
int fd;
unsigned char *buffer;
```

Figure 4-3. The "simiodemo.c" Demo Program (Cont'd)

```
int nbytes;

/*
 *      RETURNS          number of bytes read (>= 0) if sucessfull.
 *                      -1 for error and the global errno is set to the error code.
 */

{
        unsigned char *addr_ptr;
        unsigned char index;

        addr_ptr = simio_addr + 1;
        *addr_ptr = fd;
        addr_ptr++;
        *addr_ptr = nbytes;
        addr_ptr++;

        *simio_addr = S_READ;

        while (*simio_addr == S_READ)
        {
                /* Empty loop - wait for results */
        }

        addr_ptr = simio_addr + 3;

        for (index = 0; index < (*(simio_addr + 2)); index++)
        {
                *buffer = *addr_ptr;
                addr_ptr++;
                buffer++;
        }

        if (*simio_addr == 0)
        {
                return (unsigned int) *(simio_addr + 2);
        }
        else
        {
                errno = (unsigned int) *simio_addr;
                return -1;
        }
}
int
write(fd, buffer, nbytes)
int fd;
unsigned char *buffer;
int nbytes;

/*
 *      RETURNS          number of bytes written (>= 0) if sucessfull.
 *                      -1 for error and the global errno is set to the error code.
 */

{
        unsigned char *addr_ptr;
        int index;
```

Figure 4-3. The "simiodemo.c" Demo Program (Cont'd)

```
        addr_ptr = simio_addr + 1;
        *addr_ptr = fd;
        addr_ptr++;
        *addr_ptr = nbytes;
        addr_ptr++;

        for (index = 0; index < nbytes; index++)
        {
                *addr_ptr = *buffer;
                addr_ptr++;
                buffer++;
        }

        *simio_addr = S_WRITE;

        while (*simio_addr == S_WRITE)
        {
                /* Empty loop - wait for results */
        }

        if (*simio_addr == 0)
        {
                return (unsigned int) *(simio_addr + 2);
        }
        else
        {
                errno = (unsigned int) *simio_addr;
                return -1;
        }
}
int
clear_screen(fd)
int fd;

/*
 *      RETURNS          0 if sucessfull.
 *                      -1 for error and the global errno is set to the error code.
 */

{
        *(simio_addr + 1) =  fd;
        *simio_addr = S_CLEAR_DISP;

        while (*simio_addr == S_CLEAR_DISP)
        {
                /* Empty loop - wait for results */
        }

        if (*simio_addr == 0)
        {
                return 0;
        }
        else
        {
                errno = (unsigned int) *simio_addr;
                return -1;
        }
}
```

Figure 4-3. The "simiodemo.c" Demo Program (Cont'd)

```
int
pos_cursor(fd, line, col)
int fd;
int line;
int col;

/*
 *      RETURNS          0 if sucessfull.
 *                      -1 for error and the global errno is set to the error code.
 */

{
        unsigned char *addr_ptr;

        addr_ptr = simio_addr - 1;
        *addr_ptr = fd;
        addr_ptr++;
        *addr_ptr = line;
        addr_ptr++;
        *addr_ptr = col;

        *simio_addr = S_POS_CURSOR;

        while (*simio_addr == S_POS_CURSOR)
        {
                /* Empty loop - wait for results */
        }

        if (*simio_addr == 0)
        {
                return 0;
        }
        else
        {
                errno = (unsigned int) *simio_addr;
                return -1;
        }
}
int
exec_cmd(command, file1, file2, file3)
char *command;
int *file1, *file2, *file3;

/*
 *      PARAMETERS       command - a string containing the name and parameters
 *                                 of the command to execute.
 *                       file1, file2, file3 - pointers to variables to return
 *                       the file descriptors of the pipes connected to the
 *                       stdin, stdout, and stderr of the process executed.
 *                       If any pointer is NULL, that pipe is connected to /dev/null
 *                       and no file descriptor is returned.
 *
 *      RETURNS          process id (>=0) if sucessfull and the file descriptors of
 *                       the pipes are returned in file1, file2, and file3.
 *                      -1 for error and the global errno is set to the error code.
 */
```

Figure 4-3. The "simiodemo.c" Demo Program (Cont'd)

```
{
        unsigned char *addr_ptr;
        unsigned char bitmap = 0;

        if (file1 != 0)
        {
                bitmap |= 1;
        }
        if (file2 != 0)
        {
                bitmap |= 2;
        }
        if (file3 != 0)
        {
                bitmap |= 4;
        }

        addr_ptr = simio_addr + 1;
        *addr_ptr = bitmap;
        addr_ptr++;
        *addr_ptr = 255;
        addr_ptr++;

        while (*command != '\0')
        {
                *addr_ptr = *command;
                command++;
                addr_ptr++;
        }

        *addr_ptr = '\0';

        *simio_addr = S_EXEC_CMD;

        while (*simio_addr == S_EXEC_CMD)
        {
                /* Empty loop - wait for results */
        }

        if (file1 != 0)
        {
                *file1 = *(simio_addr + 2);
        }
        if (file2 != 0)
        {
                *file2 = *(simio_addr + 3);
        }
        if (file3 != 0)
        {
                *file3 = *(simio_addr + 4);
        }

        if (*simio_addr == 0)
        {
                return (unsigned int) *(simio_addr + 1);
        }
        else
        {
```

Figure 4-3. The "simiodemo.c" Demo Program (Cont'd)

```
                errno = (unsigned int) *simio_addr;
                return -1;
        }
}

int
kill(pid, sig)
int pid, sig;

/*
 *      RETURNS          0 if the process exists.
 *                      -1 for error and the global errno is set to the error code.
 */

{
        *(simio_addr + 1) = pid;
        *(simio_addr + 2) = sig;
        *simio_addr = S_KILL;

        while (*simio_addr == S_KILL)
        {
                /* Empty loop - wait for results */
        }

        if (*simio_addr == 0)
        {
                return 0;
        }
        else
        {
                errno = (unsigned int) *simio_addr;
                return -1;
        }
}

int
unlink(path)
char *path;

/*
 *      RETURNS          0 if sucessfull.
 *                      -1 for error and the global errno is set to the error code.
 */

{
        unsigned char *addr_ptr;

        addr_ptr = simio_addr + 1;
        *addr_ptr = 255;
        addr_ptr++;

        while (*path != '\0')
        {
                *addr_ptr = *path;
                addr_ptr++;
                path++;
        }

        *addr_ptr = '\0';
```

Figure 4-3. The "simiodemo.c" Demo Program (Cont'd)

```
        *simio_addr = S_DELETE_FILE;

        while (*simio_addr == S_DELETE_FILE)
        {
                /* Empty loop - wait for results */
        }

        if (*simio_addr == 0)
        {
                return 0;
        }
        else
        {
                errno = (unsigned int) *simio_addr;
                return -1;
        }
}
long
lseek(fd, offset, whence)
int fd;
long offset;
int whence;

/*
 *      RETURNS         non-negative integer indicating file pointer if sucessfull.
 *                     -1 for error and the global errno is set to the error code.
 */

{
        unsigned char *addr_ptr;

        addr_ptr = simio_addr + 1;
        *addr_ptr = fd;
        addr_ptr++;

        *addr_ptr = offset & 0xff;
        addr_ptr++;
        *addr_ptr = (offset >> 8) & 0xff;
        addr_ptr++;

        /* The following code checks to see if the compiler for this
         * processor supports 32 bit (4 byte) longs.  If so, the offset
         * parameter is used.  Otherwise only 16 bits are available and
         * the offset is sign extended to created a 32 bit integer.  This
         * code must be altered if files greater than 64K bytes long must
         * be supported with compilers which do not support 32 bit integers.
         */

        if (sizeof(long) >= 4)
        {
                *addr_ptr = (offset >> 16) & 0xff;
                addr_ptr++;
                *addr_ptr = (offset >> 24) & 0xff;
                addr_ptr++;
        }
        else  if (offset >= 0)
        {
```

Figure 4-3. The "simiodemo.c" Demo Program (Cont'd)

```
                /* Compiler does not support 32 bit longs - sign extend.      */

                *addr_ptr = 0;
                addr_ptr++;
                *addr_ptr = 0;
                addr_ptr++;
        }
        else
        {
                *addr_ptr = 0xff;
                addr_ptr++;
                *addr_ptr = 0xff;
                addr_ptr++;
        }

        *addr_ptr = whence;

        *simio_addr = S_POS_FILE;

        while (*simio_addr == S_POS_FILE)
        {
                /* Empty loop - wait for results */
        }

        if (*simio_addr == 0)
        {
                if (sizeof(long) >= 4)
                {
                        /* Compiler supports 32 bit longs -- the offset
                         * is already guarenteed to be non-negitive.
                         */

                        return ((*(simio_addr + 2) & 0xff)
                                | ((*(simio_addr + 3) << 8) & 0xff)
                                | ((*(simio_addr + 4) << 16) & 0xff)
                                | ((*(simio_addr + 5) << 24) & 0xff));
                }
                else
                {
                        /* Truncate most significant bits of absolute offset
                         * to 15 bits to guarentee that number is non-negative.
                         */

                        return (*(simio_addr + 2) & 0xff) | ((*(simio_addr + 3) << 8) &
0x7f);
                }
        }
        else
        {
                errno = (unsigned int) *simio_addr;
                return -1;
        }
}
```

Figure 4-3. The "simiodemo.c" Demo Program (Cont'd)

RS-232 Simulated I/O

Simulated I/O is a technique for reading and writing to files or devices from an emulator. A segment of emulation RAM is dedicated to the simulated I/O and shared by the emulator and the emulation system controller. To cause simulated I/O to occur, the emulator writes a command byte into the initial location of the shared segment, called the Control Address (CA). The emulation software polls this location continuously searching for commands from the emulator. When a command is recognized, the action is carried out by the emulation software on the host computer. Simulated I/O actions include open, close, read and write to a file. Since devices are treated as files in UNIX, simulated I/O to a device is similar to simulated I/O to a file.

Configuring RS-232 Lines for Simulated I/O

In order to use an RS-232 line on the UNIX host computer for simulated I/O from a emulator the line must first be configured, it can then be treated as a file. An understanding of how serial lines are dealt with in UNIX systems is an important prerequisite to configuration.

Serial Lines in UNIX Systems

When a UNIX system is booted, the initial process is called "init". This process always has process ID 1 and is the parent or ancestor of every other process running on the system. In the subdirectory /etc a file named inittab lists all the devices to be initialized by init when the system starts up. This file will contain entries for each serial line specifying the initial state of the line, whether to spawn a process for line and what to do if that process terminates. If a line is listed as requiring a process, init spawns a process called getty for that line. Getty sets the communication parameters for that line and prints the banner and login prompt on the line. When a login name is typed in, getty spawns a login process which asks for the password. If the password is correct, login forks a shell (command interpreter) process and attaches the standard input and standard output file descriptors for that shell to the /dev entries for that particular serial line.

Serial Lines Used with Simulated I/O

Using a serial line device file as a read/write file from an emulator may require a different sequence of events in the startup of that line. UNIX avoids deadlock problems by permitting race conditions to occur. Two processes can simultaneously read and write the same file, assuming both have read/write permission, without complaint from the operating system. This does not imply that there will be no problems. Two processes reading the same device file will get alternate characters. If a shell is running on a given serial line and a user attempts simulated I/O reads from that line, both the shell and the emulator will get part of the input. If both the shell and an emulator write to a serial line, the output will be interleaved. There are several ways of solving this problem, depending on the operating environment.

Serial Lines for User Terminals and Simulated I/O

The first is the situation where it is desired to generally use the serial line as a user terminal line and occasionally use it for simulated I/O. In this case the /etc/inittab file would probably contain a line to initialize the serial line for the getty, login, shell sequence of processes. The line can then be used for standard login sessions. If the user wishes to make the line available to the emulator for simulated I/O reads, the he can simply issue a sleep(1) command which will suspend the shell process for whatever period of time the user wishes. While the shell is suspended the emulator can read the serial line without interference from the shell, since the shell will issue no reads while it is suspended. Anything typed at the terminal could then be read by the emulator. If the user wished to use the line to communicate with some other device, for example, some other computer, an RS-232 ABC switch could be used to switch either the terminal or the computer to the serial line.

There is one potential problem with switching lines in this manner. The serial lines on the UNIX host computer can be configured to send a hang-up signal to the attached process if the DTR line is dropped. This is in fact the normal configuration. Switching the line from one device to another will cause DTR to be momentarily disconnected which will be interpreted as a dropped line. This will cause a hang-up signal to be sent to the shell, which will murder its siblings (sleep) and exit. When the shell exits, init will respawn a

getty process for that line which will issue a read on the serial line, creating a race condition between getty and the emulator program. This problem can be eliminated by reconfiguring the serial line, which requires powering the machine down, removing the serial line card and changing the DIP switch configuration. Alternately a special cable can be wired which will loop the modem control signals back to the serial line, making it impossible for the line to be dropped. Simply wire pins 4, 5, and 6 together and 8 to 20 on the connector to the UNIX host computer.

Dedicated Serial Lines

Another possibility is that a line is to be dedicated to communication with some external device and it is desired that the getty, login, shell sequence never occur on that line. The line is then configured to be "off" in the /etc/inittab file. In this configuration the line is initialized to standard defaults (300 baud, no parity, etc.). These defaults are nearly useless and were chosen when UNIX was an infant and have been kept around for historic reasons, there being no good reason for their existence. Another unfortunate fact is that there is no way to modify these defaults since they are hard-wired into the UNIX kernel. The most direct way to change the configuration of a port is to use the stty(1) command. The action of this command can be directed to any terminal so it can change the defaults of any terminal. Unfortunately, as soon as the stty command terminates it will close the device file at which time it will return to its default (useless) configuration. A workaround for this problem is to issue a stty command that will not terminate until the emulation process opens the device file, for example:

```
(stty -modes; sleep 100000)  /dev/tty?? &
```

In this sample command, -modes refers to the list (possibly long list) of modes the user wishes to apply to the line. The argument to sleep is the number of seconds to suspend the process. By grouping the stty and sleep commands, the affected device file will not be closed until both commands terminate. There are some drawbacks to this method. The command itself is not intuitive and will probably have to be killed manually at some later point. There is also no way to prevent another user from reading or writing the device file or changing its configuration with stty.

Another way to deal with this problem is to write a program that will configure the serial line. Since simulated I/O can spawn new processes, the program can be started and halted from within the program using simulated I/O. Using a program to configure the device file also presents an opportunity to create lock files to prevent race conditions between two processes trying to communicate with the same device. The cu(1) utility uses this technique to prevent multiple access to a single serial line.

Example Programs

Two example programs for configuring ports are presented here. The first, "useport.c" (figure 4-4), creates a lock file and configures the port. It then copies its standard input to the device file and writes any characters appearing as input to the device file to its standard output. Useport.c acts as a filter program. Communicating with a device file becomes a simple matter of starting up the useport process and communicating with its standard input and standard output.

The second example program, "setport.c" (figure 4-5), is nearly identical to useport.c. The only difference is that setport.c never reads or writes the device file, it merely configures the file and then sleeps. After the setport process is started, the device file can be opened directly for reads and writes. This method has the convenience of using a process to configure a port without the overhead of passing all the characters through an additional filter process.

Special Considerations

There are some special considerations when using simulated I/O to communicate with processes. If a process started by simulated I/O exits, there is no explicit notification of that fact. The problem is most severe when the process is opened for no wait reads. In this case a read which returns a length of zero does not indicate end of file, which is the normal way of determining that a process has terminated. When the process is started, its standard error output can be connected to a simulated I/O file descriptor. The process can indicate that it is exiting by writing a message to this file. It is also possible to use the simulated I/O "kill process" command to determine if the process is still alive. Simply send signal 0, which has no effect, a return value of 0 indicates that the process is still alive.

It is always a good idea to execute the simulated I/O "reset" command as the first simulated I/O command in a program. Simulated I/O reset will clean up any remains of a possible previous execution. This is particularly true when using simulated I/O to communicate with processes. The simulated I/O Reset command will send signal SIGPIPE to each simulated I/O process. It is important to catch this signal, do any necessary clean up and exit in any processes spawned by simulated I/O.

One final consideration is necessary if simulated I/O is to be used for machine to machine communication. Not all RS-232 devices for UNIX host computers are sufficiently buffered for reliable machine to machine communication. Single character buffered devices cannot be reliably used for machine to machine communication.

```
/* USEPORT.C
   Connect the standard input to a tty output
   and the standard output to a tty input.
   Create a lock file to prevent simultaneous
   access to the tty.
*/
#include <stdio.h>
#include <signal.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <fcntl.h>
#define  MIN    1       /* minimum chars to buffer for reads */
#define  TIME   1       /* delay for reads */

int fd, arg;
struct termio old;
char lock[256];

sig_func()
{
        /* restore stdin file mode */
        fcntl(0, F_SETFL, arg);
        /* restore terminal settings */
        ioctl(fd, TCSETA, &old);
        /* remove the lock file */
        unlink(lock);
        exit(1);
}

main(argc, argv)
int argc;
char *argv[];
{
        int i, count;
        FILE *fp;
        char c[1], buf[512], *p;
        struct termio new;

        if(argc != 2){
                fprintf(stderr,"Usage: %s ttyname\\n", argv[0]);
                exit(1);
        }
        if((fd = open(argv[1], O_RDWR | O_NDELAY)) == -1){
                fprintf(stderr,"Attempt to open %s failed\\n", argv[1]);
                exit(1);
        }
        if(isatty(fd) == 0){
                fprintf(stderr,"%s is not a tty\\n", argv[1]);
                exit(1);
        }

        /* Create a lock file */
        strcpy(lock, "/usr/spool/uucp/LCK..");
        p = ttyname(fd);

        /* this is not a nice way to skip over "/dev/" */
        p += 5;
        strcat(lock, p);
```

Figure 4-4. The "useport.c" RS-232 Sim. I/O Example

```
        if((i = open(lock, O_RDONLY | O_CREAT | O_EXCL, 0444)) == -1){
                fprintf(stderr,"Cannot create lock file %s\\n", lock);
                exit(1);
        }
        close(i);

        /* get the tty status */
        ioctl(fd, TCGETA, &old);
        new = old;

        /* set up new terminal line characteristics */
        new.c_cflag =   B9600 |         /* 9600 baud */
                        CS7 |           /* 7 data bits */
                                        /* stop bits defaults to 1 */
                        CREAD |         /* enable receiver */
                        PARENB |        /* enable parity */
                                        /* parity defaults to even */
                        CLOCAL;         /* local line */
        new.c_lflag = 0;
        new.c_oflag = 0;
        new.c_iflag =   IGNBRK |        /* ignore break condition */
                        IGNPAR |        /* ignore parity errors */
                        ISTRIP |        /* strip all chars to 7 bits */
                        IXON |          /* enable start/stop output control */
                        IXOFF;          /* enable start/stop input control */
        new.c_cc[4] = MIN;
        new.c_cc[5] = TIME;
        ioctl(fd, TCSETA, &new);

        /* set no delay on stdin */
        arg = fcntl(0, F_GETFL, arg);
        fcntl(0, F_SETFL, O_NDELAY | arg);

        /* catch all signals */
        for(i = 1; i < 24; i++)
                signal(i, sig_func);

        while(1){
                if((count = read(0, buf, 1)) == -1){
                        fprintf(stderr,"Error reading stdin\\n");
                        sig_func();
                        exit(1);
                }
                if(count){
                        write(fd, buf, count);
                }
                if((count = read(fd, buf, 1)) == -1){
                        fprintf(stderr,"Error reading %s\\n", argv[1]);
                        sig_func();
                        exit(1);
                }
                if(count)
                        write(1, buf, count);
        }
}
```

Figure 4-4. The "useport.c" Example (Cont'd)

```
/* SETPORT.C
   Set terminal characteristics.
   Create a lock file to prevent simultaneous
   access to the tty.
*/
#include <stdio.h>
#include <signal.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <fcntl.h>
#define  MIN    1       /* minimum chars to buffer for reads */
#define  TIME   1       /* delay for reads */

int fd, arg;
struct termio old;
char lock[256];

sig_func()
{
        /* restore terminal settings */
        ioctl(fd, TCSETA, &old);
        /* remove the lock file */
        unlink(lock);
        exit(1);
}

main(argc, argv)
int argc;
char *argv[];
{
        int i, count;
        FILE *fp;
        char c[1], buf[512], *p;
        struct termio new;

        if(argc != 2){
                fprintf(stderr,"Usage: %s ttyname\\n", argv[0]);
                exit(1);
        }
        if((fd = open(argv[1], O_RDWR | O_NDELAY)) == -1){
                fprintf(stderr,"Attempt to open %s failed\\n", argv[1]);
                exit(1);
        }
        if(isatty(fd) == 0){
                fprintf(stderr,"%s is not a tty\\n", argv[1]);
                exit(1);
        }

        /* Create a lock file */
        strcpy(lock, "/usr/spool/uucp/LCK..");
        p = ttyname(fd);

        /* this is not a nice way to skip over "/dev/" */
        p += 5;
        strcat(lock, p);
        if((i = open(lock, O_RDONLY | O_CREAT | O_EXCL, 0444)) == -1){
                fprintf(stderr,"Cannot create lock file %s\\n", lock);
                exit(1);
```

Figure 4-5. The "setport.c" RS-232 Sim. I/O Example

```
        }
        close(i);

        /* get the tty status */
        ioctl(fd, TCGETA, &old);
        new = old;

        /* set up new terminal line characteristics */
        new.c_cflag =   B9600 |         /* 9600 baud */
                        CS7 |           /* 7 data bits */
                                        /* stop bits defaults to 1 */
                        CREAD |         /* enable receiver */
                        PARENB |        /* enable parity */
                                        /* parity defaults to even */
                        CLOCAL;         /* local line */
        new.c_lflag = 0;
        new.c_oflag = 0;
        new.c_iflag =   IGNBRK |        /* ignore break condition */
                        IGNPAR |        /* ignore parity errors */
                        ISTRIP |        /* strip all chars to 7 bits */
                        IXON |          /* enable start/stop output control */
                        IXOFF;          /* enable start/stop input control */
        new.c_cc[4] = MIN;
        new.c_cc[5] = TIME;
        ioctl(fd, TCSETA, &new);

        /* catch all signals */
        for(i = 1; i < 24; i++)
                signal(i, sig_func);

        while(1){
                /* nap */
                sleep(1000);
        }
}
```

Figure 4-5. The "setport.c" Example (Cont'd)

Error Codes

Overview

This chapter contains:

- A table of the simulated I/O error codes.
- A description of the errors as they relate to the simulated I/O commands.

Introduction

This chapter contains a list of all simulated I/O error codes and a description of when they can occur and what condition causes each error.

Table 5-1. Simulated I/O Error Codes

ERROR CODE	ERROR NAME
03H	FILE NOT FOUND
04H	FILE ALREADY EXISTS
08H	CANNOT READ MEMORY
09H	INVALID COMMAND
0BH	INVALID ROW OR COLUMN
0FH	INVALID FILE NAME
11H	NO FREE DESCRIPTORS
12H	INVALID FILE DESCRIPTOR
16H	NO PERMISSION
17H	INVALID OPTIONS
18H	TOO MANY FILES
19H	NO FREE PROCESS ID
1AH	TOO MANY PROCESSES
1BH	INVALID COMMAND NAME
1CH	INVALID PROCESS ID
1DH	INVALID SIGNAL
1EH	NO SUCH PROCESS
1FH	NO SEEK ON PIPE
7EH	UNIX ERROR
7FH	CONTIUE ERROR

Error Code Description (By Simulated I/O Command)

General Errors

08H CANNOT READ MEMORY. The read of the command code from memory failed.

09H INVALID COMMAND. Command code is not in the range 90H through 9BH.

7EH UNIX ERROR. This error means that some UNIX system call has failed. When a system call fails, the reason for the failure is indicated in a global variable errno __ see errno (2) in the UNIX reference manual. When simulated I/O returns UNIX ERROR, the value of errno is returned in location CA + 2 (CA + 1, which contains the file descriptor, is never modified). No other error codes return errno.

7FH CONTINUE ERROR. If emulation is exited and then re-entered for a continued session, all simulated I/O files will be closed and all simulated I/O processes will no longer be executing. Any simulated I/O command that requires an open file (read, write, position cursor, etc.) issued by your program after emulation has been re-entered will get the CONTINUE ERROR error code. (Commands like open, and UNIX command will proceed with no errors.) The simulated I/O reset command will prevent additional CONTINUE ERRORs from occurring in the continued emulation session.

Open (90H) Simulated I/O Errors

0FH INVALID FILE NAME. File name length = 0.

11H NO FREE DESCRIPTORS. Too many simulated I/O files are open. The maximum is 12.

Open File Errors

03H FILE NOT FOUND. UNIX open (2) failed, and errno = ENOENT.

04H FILE ALREADY EXISTS. UNIX open (2) failed, and errno = EEXIST (create and exclusive options given).

16H NO PERMISSION. UNIX open (2) failed, and errno = EACCES.

17H INVALID OPTIONS. The open options do not contain exactly one of: read only (0), write only (1), or read-write (2). Open options not valid.

18H TOO MANY FILES. UNIX open (2) failed, and errno = EMFILE (NFILE UNIX descriptors are open by emulation).

7EH UNIX ERROR. Any error from close (2).

Close (91H) Simulated I/O Errors

12H INVALID FILE DESCRIPTOR. File descriptor indicated not open.

7FH CONTINUE ERROR. Attempt to close any file descriptor after continue. (See description under "General Errors".)

Close File Errors

7EH UNIX ERROR. Any error from close (2).

Read (92H) Simulated I/O Errors

09H INVALID COMMAND. Attempt to read from the display.

12H INVALID FILE DESCRIPTOR. File descriptor indicated not open.

7FH CONTINUE ERROR. Attempt to read anything after continuation. (See description under "General Errors".)

Read File Errors

7EH UNIX ERROR. Any error from read (2).

Write (93H) Simulated I/O Errors

08H CANNOT READ MEMORY. Cannot read buffer to write.

09H INVALID COMMAND. Attempt to write to keyboard.

12H INVALID FILE DESCRIPTOR. File descriptor indicated not open.

7FH CONTINUE ERROR. Attempt to write anything after continuation. (See description under "General Errors".)

Read File Errors

7EH UNIX ERROR. Any error from write(2).

Delete File (94H) Simulated I/O Errors

03H FILE NOT FOUND. UNIX unlink (2) failed and errno set to ENOENT.

08H CANNOT READ MEMORY. Read of file name failed.

0FH INVALID FILE NAME. File name length = 0, or unlink (2) failed (errno set to ENOTDIR).

16H NO PERMISSION. UNIX unlink (2) failed and errno set to EACCES.

7EH UNIX ERROR. UNIX unlink (2) failed for some other reason.

Position File (95H) Simulated I/O Errors

09H INVALID COMMAND. File descriptor indicates open file of type other than a UNIX file.

12H INVALID FILE DESCRIPTOR. File descriptor indicated not open.

17H INVALID OPTIONS. Start code is not 0, 1, or 2, or lseek (2) failed and errno set to EINVAL.

1FH NO SEEK ON PIPE. File descriptor indicates a pipe to a UNIX command.

7FH CONTINUE ERROR. A position file attempted after continuation. (See description under "General Errors".)

Position Cursor (96H) Simulated I/O Errors

09H INVALID COMMAND. Attempt to position cursor on file that is not a display.

0BH INVALID ROW OR COLUMN. The row number is greater than or equal to 50 rows, or the column number is greater than or equal to 80 columns (or the number of columns on the display whichever is greater).

12H INVALID FILE DESCRIPTOR. Attempt to position cursor on file that is not open.

7FH CONTINUE ERROR. Attempt to position cursor after continuation. (See description under "General Errors".)

Clear Display (97H) Simulated I/O Errors

09H INVALID COMMAND. Attempt to clear display on file that is not a display.

12H INVALID FILE DESCRIPTOR. Attempt to clear display on file that is not open.

7FH CONTINUE ERROR. Attempt to clear display after continuation. (See description under "General Errors".)

UNIX Command (98H) Simulated I/O Errors

08H CANNOT READ MEMORY. Read failed on read of command name.

11H NO FREE DESCRIPTORS. Simulated I/O descriptor table is full.

18H TOO MANY FILES. UNIX pipe (2) failed.

19H NO FREE PROCESS ID. The maximum number of processes are already active.

1AH TOO MANY PROCESSES. UNIX fork (2) failed and errno = EAGAIN.

1BH **INVALID COMMAND NAME**. Command name length is 0.

7EH **UNIX ERROR**. UNIX fork (2) failed and errno does not equal EAGAIN.

Kill Process (99H) Simulated I/O Errors

16H **NO PERMISSION**. Kill failed and errno = EPERM.

1CH **INVALID PROCESS ID**. The simulated I/O process id is unused or out of range (the simulated I/O process entry does not exist).

1DH **INVALID SIGNAL**. Kill failed and errno = EINVAL.

1EH **NO SUCH PROCESS**. Kill failed and errno = ESRCH (the UNIX process does not exist).

7EH **UNIX ERROR**. Kill failed with other error number.

Index

A
absolute byte offset (position file command), **30**
absolute file for demo program, loading the, **52**
activating the simulated I/O display, **52**
activating the simulated I/O keyboard, **53**
active processes, maximum number of, **17**
append (open file option), **22**
append option and positioning in files, **31**

B
buffer (Control Address), length of, **10**
buffer (Control Address), locating in memory, **10**
bytes to read, maximum number, **26**

C
column number (position cursor command), **33**
columns in display simulated I/O buffer, **43**
command name (execute UNIX command), **36**
command name length (execute UNIX command), **36**
compiling the simulated I/O demo program, **50**
configuring simulated I/O for the demo program, **51**
control address (CA) buffer, length of, **10**
control address (CA) buffer, locating in memory, **10**
control addresses, maximum number of, **17**
copying the simulated I/O demo program, **50**
create (open file option), **22**

D
delay, no (open file option), **22**
demo program, configuring simulated I/O, **51**
demo simulated I/O program, copying the, **50**
descriptor, file, **23**
dev/simio/display, default for standard error output, **15**
dev/simio/display, default for standard output, **15**
dev/simio/keyboard, default for standard input, **15**
display simulated I/O, clearing the display, **45**
display simulated I/O, closing, **46**
display simulated I/O, number of lines and columns, **43**
display simulated I/O, opening, **42**
display simulated I/O, positioning the cursor, **44**

display simulated I/O, writing, 43
display simulated_io emulation command, 42

E emulation ram, loading the control address into, 15
entering the measurement system for the demo program, 51
exclusive (open file option), 22

F file descriptor, 23
file descriptor for stderr (execute UNIX command), 37
file descriptor for stdin (execute UNIX command), 36
file descriptor for stdout (execute UNIX command), 37

I ID, simulated I/O process, 36

K keyboard simulated I/O, closing, 49
keyboard simulated I/O, opening, 46
keyboard simulated I/O, reading, 48
keyboard simulated I/O, reading from, 25
keyboard simulated I/O, terminating a line of input, 47

L length of path name (open command), 23
length of the control address buffer, 10
line number (position cursor command), 33
lines in display simulated I/O buffer, 43
loading the demo program absolute file, 52
locating control address into emulation ram, 15
locating the control address buffer in memory, 10

M maximum number of bytes that can be read, 26
maximum number of bytes that can be written, 27
maximum number of control addresses, 17
maximum number of open files, 17
measurement system for demo program, entering the, 51
modify keyboard_to_simio emulation command, 47

N no delay (open file option), 22

O open file option, 21
open files, maximum number of, 17
open pipe specification (execute UNIX command), 36
opening display simulated I/O, 42

P path name (open command), 23
pipe, open specification (execute UNIX command), 36
positioning in files opened with the append option, 31

process ID (execute UNIX command), **36**
processes (active), maximum number of, **17**
protocol (general), **10**

R read command, maximum number of bytes that can be read, **26**
read only (open file option), **22**
reading from keyboard simulated I O, **25**
real-time, **9**
relative byte offset (position in file command), **31**
reserved file names (stdin, stdout, and stderr), **15**
reserved files, attempting to delete, **29**
RS-232 simulated I/O, **69/77**

S signal to send to process (kill command), **38**
SIGPIPE, **39**
simio_ca_xxx, symbols associated with control addresses, **15**
simulated I/O commands, list of, **19**
simulated i/o control address? (configuration questions), **14**
simulated I/O demo program, copying the, **50**
simulated I/O demo, configuring, **51**
simulated I/O process ID (execute UNIX command), **36**
softkeys (when keyboard simulated I/O is active), **48**
special files, attempting to delete, **29**
starting location (position in file command), **31**
stderr file descriptor (execute UNIX command), **37**
stderr, reserved file name, **15**
stdin file descriptor (execute UNIX command), **36**
stdin, reserved file name, **15**
stdout file descriptor (execute UNIX command), **37**
stdout, reserved file name, **15**
suspend softkey, **48**

T terminating an input line (keyboard simulated I/O), **47**
truncate (open file option), **22**

U using display simulated I/O, **42**

W write command, maximum number of bytes that can be written, **27**
write only (open file option), **22**

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

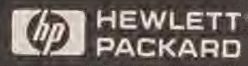
Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Hewlett-Packard
Printed in the USA



User's Guide

Symbolic Retrieval Utilities

53

79

83

HP 64700 Operating Environment

Symbolic Retrieval Utilities

User's Guide

HP Part No. B1471-97007
Printed in U.S.A.
March 1992

Edition 2

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

© Copyright 1991, 1992, Hewlett-Packard Company.

This document contains proprietary information, which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and in other countries.

Hewlett-Packard
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901-2197, U.S.A.

RESTRICTED RIGHTS LEGEND Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in subparagraph (C) (1) (ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013. Hewlett-Packard Company, 3000 Hanover Street, Palo Alto, CA 94304. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Printing History

New editions are complete revisions of the manual. The date on the title page changes only when a new edition is published.

A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual revisions.

Edition 1 B1471-97004, July 1991
Edition 2 B1471-97007, March 1992

What are Symbolic Retrieval Utilities (SRU)

SRU provides a mapping between symbols and addresses for some HP 64000 products, such as emulation. The symbols may be user-symbols (such as labels in assembly language, procedure names in C, and packages in ADA) or compiler generated symbols (such as loop-labels from the HP Advanced Cross Language System (HP-AxLS) C compiler).

SRU consists of three supported utilities, an unsupported utility, and a library that is loaded into products such as emulation. The three supported utilities provided by SRU are srubuild, sruclean, and sruprint. The unsupported utility included in the SRU software provided to the user is sruaccess.

Srubuild builds a symbol database and is a replacement for the edbuild utility for some emulators.

Sruprint prints the symbols in a symbol database and is a replacement for the edbprint utility.

Sruclean removes or cleans up the datebase that is created for each absolute file.

Sruaccess allows interactive examination of virtually all data within the database through the use of the SRU library.

The architecture of SRU requires several other executable and text files, but these files are not directly visible to the user.

In This Book

Chapter 1 **Using Srubuild.** This chapter how to use the srubuild utility to build a symbol database.

Chapter 2 **Using Sruclean.** This chapter how to use the sruclean utility to remove or clean up a symbol database.

Chapter 3 **Using Sruprint.** This chapter how to use the sruprint utility to convert the contents of a SRU symbolic datebase file to a readable format and print the contents to stdout.

Chapter 4 **Using Sruaccess.** This chapter how to use the sruaccess utility to interactively examine data in an SRU database.

Appendix A **Finding More Information**. This appendix lists the online man pages for commands and shows you how to access them.

Contents

1 **Using Srubuild**

What Is In This Chapter? 9
What Is Srubuild? 10
Incremental Builds 10
Partial Builds 11
HP-AxLS and SRU 12
Without SRU 12
With SRU 12
AxDB and Emulators With SRU 12
SRU/EDB Compatibility 12
Symbols in SRU 13
Arrangement of Symbols 13
Language Dependencies 15
Procedure Special Symbols 16
Symbol Attributes 17
Symbol Types 18
Symbol Levels 19
Segment Symbols 20
Language Sections Versus Segments 20
Symbol Tree. 21
Linker's View. 22
Physical View. 22
Unnamed Block Symbols 23
Renaming of Symbols 24
Maximum Symbol Length 24
Entering Symbols Using --EXPR-- 24
The Values Referred To 25
HP64KSYMBPATH and cws 27
Procedure Special Symbols 28
Segment Symbols 28
Printing of Symbols (in Trace Lists) 28
Selecting High/Lowlevel Symbols 29
File Names 30
UNIX File Names 30
Non-UNIX File Names 31

Search Algorithm for Locating File Name Symbols 31
How SRU File Name Mapping Works 33
Advantages 33
The HP64_DEBUG_PATH Variable. 34
File Name Translation Table 35
Algorithm for Mapping File Names 36
Examples of Use 38
Scenario 1 38
Scenario 2 38
Scenario 3 39
Scenario 4 40
Scenario 5 41
Messages Generated By SRU 42
Error Messages 42
IEEE-695 Specific Error Messages 52
Warning Messages 54
IEEE-695 Specific Warning Messages 55
How To Use The Srubuild Command 58
Examples Using Srubuild 61

2 Using Sruclean

What Is In This Chapter? 63
What Is Sruclean? 64
How To Use The Sruclean Command 65
Examples Using Sruclean 66
Example 1: 66
Example 2: 66
Example 3: 66
Example 4: 66

3 Using Sruprint

What Is In This Chapter? 67
What Is Sruprint? 68
How To Use The Sruprint Command 68
Examples Using Sruprint 69
Example 1: 69
Example 2: 69
Example 3: 69

4 Using Sruaccess

What Is In This Chapter? 71
What is Sruaccess? 72
How To Use The Sruaccess Command 73
Sruaccess Commands 73
Data-entry Formats 76
Interactive Versus Batch Mode 77
Processor ID Names 77
Address Types 77

A Finding More Information

Index

Using Srubuild

What Is In This Chapter?

The information in this chapter includes:

- What Srubuild Is.
- HP-AxLS and SRU.
- Symbols in SRU.
- File Names .
- Printing of Symbols in Trace Lists.
- How SRU File Name Mapping Works.
- Messages You May Get While Building an SRU Data Base.
- How To Use The Srubuild Command.
- Examples Using Srubuild.

What Is Srubuild?

Srubuild builds a symbol database from an object module. The object module is typically built by a compiler, assembler and linker. The symbol database consists of several files that are stored in a single subdirectory. The subdirectory name is based on the name of the absolute file. For example, the subdirectory name for an absolute file named "myprog.X" will be "myprog.Ys".

Database files with an extension ".GY" or ".LY" are considered private to SRU. Users should not remove or modify any of these files except as occurs as a result of action from one of the SRU utilities.

The srubuild utility has been designed to read in only part of the symbol or executable file at a time and write out the database as the symbols are read in. The database consists of a set of files that are stored in a single subdirectory. One file contains all the global symbols; this would be all the symbols in the linker symbol file (.L file) for HP 64000 format files.

There is also a database file for each set of the local symbols; this is all the symbols in a single assembler symbol file (.A file for HP 64000 format files).

Incremental Builds

Srubuild is capable of doing incremental builds for object formats that contain sufficient information to support this feature. If one source file in a multi-file program is modified and the program is then compiled and linked, the symbol information for the files that were not modified is retained from the previous srubuild; only the modified symbols need to be rebuilt. This feature reduces the amount of time required to build a symbol database when changes are made in the source files. The HP/MRI IEEE-695 format and HP 64000 format are the only currently supported formats that contain sufficient information for this feature.

The srubuild utility and SRU library benefit from previous builds by using a process called "incremental build". The SRU global symbol file and each local symbol file contains a creation date. When an absolute file is updated, the dates in the SRU database files are compared to dates in the absolute file. Any symbol files that were not modified when the absolute was updated can continue to be used without modification. The global symbol file

will be updated whenever the absolute file changes, along with any SRU local symbol files that are affected.

The ability of this incremental build methodology to work is dependent on having correct dates in the absolute file. The HP/MRI IEEE 695 file format contains a date for every module in the absolute file. Language tools which place this date correctly in the IEEE file (such as the HP-AxLS language tools) will work correctly with incremental builds. The HP 64000 absolute file formats do not contain creation dates. In this case, the .X, .L and .A file modification dates are used as creation dates. Incremental builds work as long as file modification dates are not changed. Changing the modification date will have the same effect as recompiling the module, and will cause the symbol information to be rebuilt for that file.

Partial Builds

The srubuild utility and SRU library are also capable of "partial builds"; that is, building only part of the database. If you choose to use the srubuild utility you can specify which modules to build or ignore. The edbuild utility had a similar option. An important extension available in SRU is the ability to build additional parts of the database at a later time. For example, you can ask the srubuild utility to create symbol information for only the modules of interest. If, during emulation, you need to access a symbol that is not in the SRU database, SRU will automatically create the missing symbol information.

These abilities cause some side effects in your emulation tools. For example, if you display (symbols on) a trace list that contains addresses from files that have not been built, there will be a pause in the trace display while the symbol information is added to the SRU database. There will be a message on the status line informing you that symbol information is being updated, and the status line will report progress every few seconds. Another side effect is that if you build the database from within emulation (rather than using the srubuild command in your makefile) only the global file will be built; local files will be built on demand. If srubuild is invoked from the command line, both local and global information is built.

HP-AxLS and SRU

Without SRU

Before SRU, it was necessary to compile and link with the -h option when using an HP-AxLS (Advanced Cross Language System) compiler (such as cc68000). When using the -h option, the HP-AxLS compiler would generate files (file.L, file.X, file.A) needed by EDB.

Note that the HP-AxLS compilers let you compile all the files with -h, then link either with -h or without, letting you create two different absolute files (with different OMFs) without having to recompile the source files.

With SRU

SRU will read the HP/MRI IEEE-695 OMF that the HP-AxLS generates without the -h option. Further, better symbol information will be available. Therefore, it is preferable to remove the -h option on your compile lines.

AxDB and Emulators With SRU

Note that the Advanced Cross Debug (AxDB) tools do not require the -h option, so if you used a debugger and an emulator using EDB, you had to do one link with the -h and another link without the -h option. Now that you have an emulator using SRU, you may use the same absolute file with the debugger and with the emulator.

SRU/EDB Compatibility

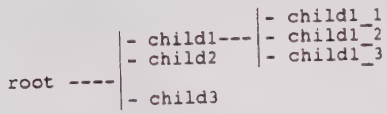
Since SRU and EDB create different databases, it is permissible to link with the -h option and without the -h option, and then use srubuild and edbuild. Although this is an option, there is probably no reason that you would ever need to use it.

Symbols In SRU

Arrangement of Symbols

Symbols are arranged in a 'tree' structure that mimics the natural scoping of the user's source language as much as possible. All emulation references to symbols - both input from the keyboard and output displays - make use of the tree structure to show the scoping of symbols and to disambiguate symbols that have the same name but different scopes.

Each absolute file has its own symbol tree. The exact entries in the tree depends on the language used, but in general the tree looks like:

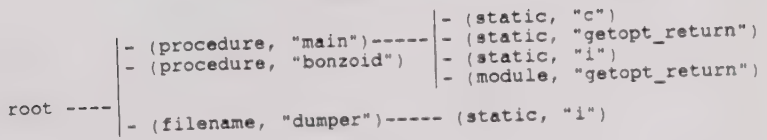


Each entry in the symbol tree has a type and a name. The types define such attributes as "procedures", "tasks", etc. They do not indicate language types, e.g. "int" or "char *". See "Symbol Attributes", later in this chapter for more information.

The names are an ASCII string, such as "main" or "sub_program_1".

Sometimes a child symbol can appear to be in two places. For example, consider a global symbol; it is considered to be a child of "root" and a child of the file in which it is defined. In this case consider the symbol to be accessible in both places (for both input and output).

You can uniquely identify any entry in the symbol tree with a combination of type-name pairs, just as UNIX file names are uniquely identified by their paths from /. For example, consider the following tree, where each entry is shown as (<type>, <name>):



The type "static" refers to a symbol with a fixed address, and is not one of the attributes described above (procedure, module, task, file name, segment, procspecial (ENTRY/EXIT), or source-reference).

In this tree, there are two entries with the same name (static, "i"), but they are unique because their full paths are different:

symbol 1: (procedure, "main"), (static, "i")
symbol 2: (filename, "dumper"), (static, "i")

The full path of a symbol consists of the names of the entries separated by dots (or colons if the preceding symbol was a filename). The "root" is never displayed. The types are not usually displayed or entered as part of the name unless necessary; for example:

symbol 1: main.i
symbol 2: dumper:i

Note that this is equivalent to, but a lot faster to type in than:

symbol 1: main(procedure).i(static)
symbol 2: dumper(filename).i(static)

In the case where two symbols with the same parent have the same name, the type information is necessary, and is added as part of the ASCII string which represents the symbol. For example:

symbol 1: main.getopt_return(static)
symbol 2: main.getopt_return(module)

Language Dependencies

SRU attempts to reconstruct the user's view of user symbol space. However, it is limited by the information that the Object Module Format (OMF) can represent. For example, a common C language tree might look like:

```
                                                                  |-- (static, "j")
root-|- (filename, "main.c")----|- (procedure, "main")---|
                                                                  |-- (static, "index")
     |                          |- (static, "global")
     |- (filename, "dump.c")----- (static, "i")
     |- (static, "global")
```

However, for an OMF which is task-based (such as the 80386 OMF), the linker enforces the creation of "tasks". For the 80386, the above tree would look like:

```
                                                                  |-- (static, "j")
     |- (module, "main")----|- (procedure, "main")---|
                                                                  |-- (static, "index")
root-|                      |- (static, "global")
     |                      |- (filename, "main.c")
     |
     |- (module, "dump")----|- (static, "i")
     |                      |- (filename, "dump.c")
     |
     |- (static, "global")
```

Note that in the 80386 OMF the global symbols would be "main", "dump" and "global" instead of "main.c", "dump.c" and "global". Also note that if you want to refer to the variable "j" in the procedure main, you would enter a different command. (Refer to "Entering Symbols Using --EXPR--", later in this section for more information.)

```
main.c:main.j    # non-80386 OMF reference to "j"
main.main.j      # 80386 OMF reference to "j"
```

If you are unsure of what your "language tree" looks like, you can use the sruprint program to print out portions of your tree.

Procedure Special Symbols

All symbols of the type "procedure" have zero or more children of the type "procspecial". These symbols contain information about the entry and exit points of the procedure. They each contain the address of the exit or entry point of the procedure. In EDB, these symbols existed but were hidden from the user. In SRU, they will show up on a "display symbols" listing, and can be entered from the keyboard like any symbol, or you can continue to use the implied method from EDB.

They have names of the form:

ENTRY	if this is present, it means that the procedure had exactly one entry point. This is the most common case; FORTRAN and ADA can have multiple entry points as of this writing.
EXIT	if this is present, it means that the procedure had exactly one exit point. This is common when there is a large amount of code generated for an exit (for example, popping the stack or resetting an interrupt state), or the procedure has been compiled for debugging. Note that if a C function has multiple return statements it may or may not have multiple exit points; the compiler makes this decision.
ENTRY0, ENTRY1, ...	These are present when a procedure has multiple entry points.
EXIT0, EXIT1, ...	These are present when a procedure has multiple exit points.
TEXTRANGE	This symbol has the range of the code associated with the specified procedure. This may be different from the range of ENTRY through EXIT if the compiler generates code (such as a code-space optimized subroutine) which is after the EXIT.

DATARANGE — This symbol has the range of the code plus data associated with this procedure. This is different from TEXTRANGE in the case where the compiler generates data associated with a procedure that is after the last section of code.

Consider the following two procedures:

```
procA(a, b)
{
   if (a == b)
      return a;
   return b;
}

procB(a,b)
{
   int rvalue;
   if (a == b)
      rvalue = a;
   else
      rvalue = b;
   return rvalue;
};
```

There will be the following symbols in the SRU (depending on the compiler):

```
(procedure, "procA")----|--(procspecial, "ENTRY")
                        |--(procspecial, "EXIT0")
                        |--(procspecial, "EXIT1")

(procedure, "procB")----|--(procspecial, "ENTRY")
                        |--(procspecial, "EXIT")
```

Symbol Attributes

Each symbol in SRU has a "type" and a "level". The "types" are used when necessary to disambiguate between two symbols of the same name but of different type. You may specify the type of a symbol on input, and the output routines will display the type when necessary. You may also select the "level" of symbols that will be displayed.

Symbol Types

static — Static symbols - includes global variables (global to a specific task and/or program). The static does not mean that the symbol declaration was prefaced with the C static keyword. It means that it has a logical address which will not change.

procedure — Procedure (or function) symbols. Also used for code blocks in C. For example; when

```
m()
{
   static int alpha;

   if (alpha == 0)
   {
      static int o;
   }
}
```

occurs in a C file, the following tree is built:

```
(procedure, "m")--|-(static, "alpha")
                  |-(procedure, "BLOCK_1")--|-(static, "o")
```

Refer to "Unnamed Block Symbols" later in this section for more information on block symbols.

filename — Source file name symbols. These are "symbols" which actually define the name of a file (such as the ".c" files for C). Source references occur only under filename symbols.

module — Module symbols. For 80386 C, these names are derived from the source file name. For Ada, they are packages. Other language systems may permit the user to explicitly name these.

task	Denotes task symbol. Task symbols are specific to a multi-tasking environment and therefore have a range defined by that task. The meaning of the range may vary with processor and language system used.
procspecial	Special symbol names are used to denote language-system symbols not defined by the user (refer to "Renaming of special Symbols" later in this section for more information). Any symbol of this type is always the direct child of a renaming of symbol.
fsegment	This is currently used only in the 80386 environment to hold code or data fsegments that are in the Global Descriptor Table (GDT). The names are made up and only information about their address range is available.

Symbol Levels

highlevel	This indicates that the symbol was emitted by a "high level" in the tool chain (that is: symbols that were available from the compiler).
lowlevel	This indicates that the symbol was generated by a compiler or is an assembler symbol.

The following truth table defines the symbol level conditions and their meanings.

hilevel	lowlevel	Meaning
0	0	Cannot happen
0	1	This symbol is a low level symbol
1	0	This symbol is a high level symbol
1	1	Insufficient information to determine what this is (that is, it might be high or low level)

Refer to "Selecting High/Lowlevel Symbols" later in this section for more information.

Segment Symbols

Depending on the OMF, "segment" symbols will be generated. The purpose of the "segment" symbols is to provide a convenient method of relating an assembler listing and a linker listing with what is displayed on the screen.

An assembler listing shows addresses as offsets from the start of a 'segment'. The linker listing shows the start of each segment. Without the SRU 'segments', you would have to subtract the linker's "start of segment" to obtain an offset that you could use with your assembler listing.

Language Sections Versus Segments

A language "section" consists of all identically-named segments which are contiguous in memory; a "segment" starts with each source file.

To show this difference, consider the examples on the following pages.

The set of tables for the linker's view and the physical view, that follow, all refer to the following four symbols:

fileA and fileB are files;
symA is a symbol which is defined within fileA (for example, it might be a variable outside of any renaming of scope);
symB is a symbol defined in fileB.

Further, the tables assume the following HP-AxLS linker commands:

```
SECT  PROG  $1000        # Start of full PROG section
SECT  DATA  $3000        # Start of full DATA section
LOAD  fileA,fileB        # fileB's PROG section is placed
                         # immediately after fileA's PROG
                         # section, and fileB's DATA section will
                         # immediately follow  fileA's DATA
                         # section
```

Symbol Tree. The symbol tree associated with this example, including segment information, is:

```
                                  |-(static, "symA")-|-(segment, "PROG")
          |-(filename, "fileA")--|-(segment, "PROG")
          |                       |-(segment, "DATA")
root ---|
          |-(filename, "fileB"(--|-(static, "symB")-|-(segment[0], "PROG")
                                  |-(segment, "PROG")
                                  |-(segment, "DATA")
```

Note

The "segments" are shown as being in the tree only for the sake of making this clearer; they are not in the tree, in that you cannot enter a symbol named

fileA:PROG

and get anything back (other than "symbol not found"!), and they do not show up in the "display symbols" command.

Linker's View. This shows how the two segments (sections) defined above (PROG and DATA) are loaded into memory:

Full sections	PROG	DATA
Start address	1000	3000
	(fileA's PROG @ 1000	(fileA's DATA @ 3000
	fileB's PROG @ 1101)	(fileA's DATA @ 3053)
End address	1200	3098

Physical View. This shows how the symbols and files are loaded into memory. The "offset from start of segment" is useful to know when you are looking at a linker listing and at an assembler listing. Note that while the files have DATA segments, the symbols do not. (This is typically defined by each compiler/assembler/linker tool chain.)

symbols	fileA	symA	fileB	symB
	PROG			
absolute start address offset from start of section	1000 0	1030 30	1101 101	1154 154
absolute end address offset from start of section	1100 100	1036 36	1200 200	1167 167
	DATA			
start absolute address offset from start of section	3000 0	---- ----	3053 53	---- ----
absolute end address offset from start of section	3052 52	---- ----	3098 98	---- ----

Unnamed Block Symbols

Consider the following code:

```
main()
{
   static int outside_static;

   if (j == 0)
   {
      static int inside_static;
   }
}
```

You can easily get access to the variable "outside_static" by using the name:

main.outside_static

but you get access to the variable "inside_static" as follows:

SRU will create a name for each "unnamed code block", so that any variables declared in it are accessible. The format for the name depends on the OMF. Some OMFs will tell the line number where the code block starts. In that case, the name created for the unnamed code block will be in the form:

BLOCK_<line_number>

If the OMF tells only that you are entering an unnamed code block, the format will be:

<ordinal number>_BLOCK

Where <ordinal number> starts with "1" and increments for each encountered code block. To discover which code block a given <ordinal number>_BLOCK is related to, use "display symbols" to look at the addresses of source references, then the address associated with the <ordinal number>_BLOCK.

Some language tools will create their own name for "unnamed code blocks". In that case, SRU will pass on the created name. Do a "display symbols <procedure name>" to see what they look like when you encounter this case.

Renaming of Symbols

Some languages (notably Ada) allow multiple symbols with the same names.

For example, Ada allows a procedure named "multiply" with three parameters and a different procedure named "multiply" with two parameters; the correct procedure is called based on the number of parameters in the invocation.

However, SRU does not allow two symbols of the same "type" to have the same name (where "type" is "procedure", "module", "procspecial", etc.). So, SRU will rename the symbols by adding a unique suffix to each instance. To determine what symbols were renamed, use the "display symbols <symbol>" command and look at the address ranges.

Maximum Symbol Length

SRU does not place any limit on the number of characters that a symbol (a language symbol or a filename) may have. However, HP/MRI IEEE-695 has a maximum length of 127 characters, and OMF-386 has a limit of 40 characters. Further, each tool chain (typically the assembler or compiler and/or linker) places a constraint on the maximum number of characters a symbol may have.

For HP-AxLS, the assembler has a significance limit of 31 characters; the compiler 30 characters for external symbols, and 255 characters for non-external symbols. All SRU utilities are compatible with the UNIX "long filenames" system.

Entering Symbols Using --EXPR--

Entering symbols from the command line is done using the "--EXPR--" prompt softkey. The command syntax for entering a symbol is:

```
[.|:][<file:>|<ident.>[(<type>)]]*{<ident>[(<type>)]|<file:>}
```

optionally, this is followed by:

```
procedure {entry_exit_range | text_range}
```

Refer to "Renaming of special Symbols," later in this section for more information.

For comparison, EDB's command syntax was:

```
[:][<file:>][<ident.>]*<ident>
```

Note that this is expressed in a common UNIX notation:

```
<string>                - a non-literal string
[<anything>]            - anything between brackets can be omitted
{<string1>|<string2>}   - string1 or string2 is valid as input
[<string1>|<string2>]   - either string1, string2, or nothing is valid
[<anything>]*           - anything may be repeated indefinitely or omitted
```

Some examples:

```
globalvar          # a global variable - but see HP64KSYMBPATH later in
                   # this section

:globalvar         # The leading colon forces "globalvar" to refer to a
                   # global variable instead of a local variable. A
                   # leading dot means the same thing.

main.c:index       # anything preceding a colon (up to a previous
                   # colon) is part of a filename

file1.c:file2.c:   # file2.c has actual code but is an include file
                   # reference in file1.c

file1.c:"file2.c"  # a variable named "file2.c" in file file1.c  Use
                   # quotes to surround any "special characters" (dots
                   # and colons) to prevent them from being used as
                   # separators.

"a:b.c":alpha      # the variable named alpha in the file named a:b.c -
                   #  again note the use of quotes to prevent inter-
                   # pretation of the colon after the 'a'

.procedure.block1.block2.static_variable  # a global procedure....

 :main(procedure)  # a global procedure - but there is another
                   # global symbol named 'main' which is of a
                   # different type

 :package1."file.c":  # the file "file.c" which is a child of the global
                      # symbol "package1". If a file is a child of a
                      # non-file symbol, the entire filename
                      # must be surrounded in quotes.
```

The Values Referred To

The following rules determine what address is referred to when entered using either the --EXPR-- or --SYMB-- softkeys:

1. If a procedure is entered, the default range is the ENTRY/EXIT range. If the ENTRY/EXIT range does not exist, the TEXTRANGE is used (and you will receive a warning on the status line). If the TEXTRANGE does not exist, the range of the symbol itself is used (and you will receive a warning on the status line).

2. If a filename is entered, the default range is the segment named "PROG". If there is no such segment, the segment named "prog" will be used. If that does not exist, the first segment with attribute "code" will be returned (if the OMF supports it). Otherwise, no symbol is returned.

3. If a line number is specified, the default range is the first through last address associated with that line number.

4. If math is performed on a symbol (including the three outlined above), the starting address associated with that symbol is used unless the "end" softkey is present.

5. If the symbol is in a context that cannot use a range (such as "display memory at <symbol>"), the start address associated with the symbol is used.

Examples:

```
trace_on_range file.c:              # trace addresses between the start and end addresses
                                    # of the "PROG" segment of file.c (or "prog" segment,
                                    # or first segment with type "code" found)
trace_on_range proc(procedure)      # trace addresses between the start and end addresses
                                    # of the ENTRY/EXIT range of proc (or, if the OMF does
                                    # not support ENTRY/EXIT range, the TEXTRANGE)
trace_on_range symbolA              # trace addresses between the start and end addresses
                                    # of the symbol "symbolA"
trigger on symbolA                  # trigger on the start address of the symbol "symbolA"
trigger on symbolA end              # trigger on the end address of the symbol "symbolA"
trigger on symbolA + 4              # trigger on 4 plus the start address of the symbol
                                    # "symbolA"
trace_on_range proc text_range      # trace addresses between the start and end addresses
                                    # associated with the TEXTRANGE of procedure 'proc'
```

HP64KSYMBPATH and cws

SRU has symbol-searching capability. Further, it has the ability to explicitly set a "current working symbol" (cws), which allows you to refer to symbols relative to the cws (just like a UNIX 'cd' command allows you access to files below your current working directory).

When the shell variable HP64KSYMBPATH is set (and exported) to be a blank-separated list of symbols, a "search list" is set. When a symbol is entered without the leading colon or dot (which forces it to be global), the following happens:

1. The current working symbol (if there is one) is prefixed to the entered symbol; if the resulting symbol exists, that will be the symbol used.

2. For each entry in HP64KSYMBPATH:

   a. prefix the entry with the entered symbol. If the symbol exists, that is the symbol to use.

   b. Otherwise, remove the last entry in the HP64KSYMBPATH's symbol and repeat the previous step (refer to the following examples - they make the meaning of this statement clearer.)

Example:

```
set HP64KSYMBPATH=".file1:proc1 .file2:proc2:code_block_1"
cws :omega.c:   # cws is the file "omega.c"
hello.stat      # the entered symbol


look for symbol :omega.c:hello.stat          # prefix with cws (assume symbol not found)
look for symbol :file1:proc1.hello.stat      # prefix with first entry in HP64KSYMBPATH
look for symbol :file1:hello.stat            # remove one element & prefix
look for symbol :file2:proc2.code_block_1.hello.stat  # prefix with second entry in
                                                      # HP64KSYMBPATH
look for symbol :file2:proc2.hello.stat      # remove one element & prefix
look for symbol :file2:hello.stat            # remove one element & prefix
```

The current working symbol can always be changed with the "cws" command. Some emulation products may supply a softkey-method of changing it, but "cws <symbol>" is always available.

Further, the command "pws" (print working symbol) is also available. It displays the current working symbol on the status line.

Procedure Special Symbols

When you have entered a procedure, SRU provides a special syntax to access the ENTRY and EXIT symbols that are associated with a procedure. The syntax is:

<symbol> procedure {entry_exit_range | text_range}

Using the entry_exit_range will cause SRU to return all ENTRY and EXIT symbols related to the procedure <symbol>. For speed of prompting, the special syntax is available even when you have not entered a procedure; but, after you hit <RETURN>, if the <symbol> is not a procedure an error is generated.

If you use "text_range", the addresses associated with TEXTRANGE will be returned by the parser.

Segment Symbols

When you have entered a filename, you may use the syntax:

<filename> segment

Where is a string (such as "axls_named_segment") with or without quotes. In EDB, there were only 3 segments: "PROG ", "DATA ", and "COMM ". For compatibility, if SRU encounters a segment named "PROG " but no segment named that exists, it will search for a segment named "PROG" or "prog".

If you enter a <filename> with no "segment ", the segment named "prog" will be used. For OMFs which support 'typing' of segments ("code", "data", or "unknown"), if no segment named "prog" exists, the first segment of type "code" will be used.

Printing of Symbols (In Trace Lists)

In general, when a symbol is displayed in the trace list it looks much like it would look if the user typed the symbol in.

SRU will display as much of the symbol as possible. If necessary, the following truncation rules (from last to be truncated to first to be truncated) apply:

1. Sign of the offset (+/-) truncated on the right.

2. Last part of symbol name truncated on the right.

3. If the next to last part of the name exists and is not a filename (most closely enclosing scope), it is next, truncated on the right.

4. Basenames of parts of symbol name that are filenames truncated on the right.

5. Last part of segment name truncated on the right.

6. Remaining name parts that aren't file names truncated on the right as individuals but taking the most closely enclosing ones first (truncating left to right).

7. Remaining names from any parts that are filenames.

The basename is the last name on the right for UNIX filenames. UNIX filename components are separated by slashes. Remaining names are taken in right to left priority for UNIX filenames.

The segment name will be separated from the symbol name by a '|'. File names will be separated from the next name part by a ':'. The name parts that aren't filenames will be separated from each other by a '.'. A ':' or '|' that occurs as the right most character will be output, but a '.' in this position will be suppressed. If NULL names are encountered, they are replaced by "??" and an appropriate separator is added.

Selecting High/Lowlevel Symbols

Each product will have a command to select either high, low, or both high and low level symbols for display. When this command is given, only the specified level of symbol will be displayed in trace lists, "display symbols" commands, etc. Note, however, that any level of symbol may be entered at any time.

File Names

UNIX File Names

SRU (for OMF's that use file names) maintains a list of all files used in making an absolute. When you enter a filename without its path (e.g. "util.c" instead of "/users/mike/util.c"), a search is made of the filename list. If there is only one file named "util.c", it does not matter what directory you are in - you refer to that file. If there is more than one "util.c" used to make the absolute file, the one in your current directory will be chosen (if that was one of them), or neither will be chosen (yielding a "symbol not found") for the case where neither of the two files are in your directory.

The exact search algorithm used is described in detail at the end of this section.

A relative filename works the same way. Examples:

Files in SRU's file list:
/users/mike/src1/util.c
/users/mike/src1/globs.c
/users/mike/src2/util.c

Current directory: /users/mike

```
Input filename     filename referenced
globs.c            /users/mike/src1/globs.c    # no conflict
util.c             - symbol not found -        # conflict: which 'util.c'?
src1/util.c        /users/mike/src1/util.c     # no conflict; the "src1" was sufficient
                                               # info to disambiguate between the two
```

Current directory: /users/mike/src2

```
Input filename     filename referenced
util.c             /users/mike/src2/util.c       # current directory used to disambiguate
src1/util.c        /users/mike/src1/util.c       # again "src1" sufficient
```

As a further convenience (especially for those uploading files from a system which only supports upper-case filenames), if the search for a filename fails, SRU will search again; this time case-insensitively.

Non-UNIX File Names

As long as the OMF permits, the filename that you wrote the file with will be used in SRU. For example, if you wrote and compiled the files on a VAX/VMS system, then transferred the files to UNIX, you could refer to your VMS filenames if you used the HP/MRI IEEE-695 file format.

Any filename may be entered as long as:

1. Any characters that are special to the parser (e.g. ":", "(", "+") must be escaped by quoting them.

2. the filename is followed by a non-escaped colon.

For example:

```
hello.c:dolly.c:          # two files: the file "dolly.c" as included by "hello.c"
"hello.c:dolly.c":        # one file named "hello.c:dolly.c"; the colon is escaped
                          # by the quotes
"vaxa::user$disk:[del.bozo]util.c":       # the colons are escaped
"dir1\dir2\file.c":                       # the backslashes are escaped
```

Search Algorithm for Locating File Name Symbols

As previously mentioned, SRU maintains a list of all files used in making the absolute file. When you enter a file name, the list of files is searched, looking for a match. If a match cannot be found, SRU will cause an error message to be printed. The following steps describe the sequence of events for finding a match.

Step 1. Search for an exact match. Match found?

Yes	Search ends. (This will always work if the file name is typed exactly as it appears within the OMF file).
No	Go to step 2.

Step 2. Search again using a truncated search looking for any file name ending with the same file name entered on the command line. Exactly one match found?

Yes	Search ends.
No	Go to step 3.

Step 3. More than one file name found in step 2?

Yes	Go to step 4.
No	Go to step 5.

Step 4. Prefix the current working directory to the file name entered on the command line and search for an exact match. (This step provides backwards compatibility with EDB for disambiguating multiple file name symbols by using the current working directory.) Match found?

Yes	Search ends.
No	Go to step 8.

Step 5. Search for a match using a case-insensitive search. (This step provides more flexibility for entering case-insensitive VMS and MS-DOS file names.) Exactly one match found?

Yes	Search ends.
No	Go to step 6.

Step 6. More than one match found in step 5?

Yes	Go to step 8.
No	Go to step 7.

Step 7. Search for a match using a case-insensitive, truncated search looking for any file name ending with the same file name entered on the command line. Exactly one match found?

Yes	Search ends.
No	Go to step 8.

Step 8. Two or more files matching?

Yes	Message: Ambiguous file names.
No	Message: Symbol not found.

Note

VMS version numbers and version delimiter (;) are only used in comparisons if the file name entered on the command line is specified with a version number and version delimiter.

How SRU File Name Mapping Works

SRU needs to access source and symbol (HP OMF .A) files in order to display source references and to build the local symbols database. The path names of these files are extracted from the absolute (OMF) file when the global symbols database file is created. Normally these file names reflect the correct paths to these files unless the files have been moved or the absolute file was created on a different host. For example, if a program is compiled and linked on a remote VAX/VMS system, the absolute (OMF) file will typically contain embedded VMS path names. In the past, when files are moved, the user was required to change the file names within the absolute (OMF) file to reflect the new paths. However, SRU has implemented a feature which allows the user to specify the location of user files and still retain the original path names within the absolute file.

Advantages

- Preserves the original file names within the OMF, allowing the user to see and reference files by their original name.
- Allows UNIX, VMS and/or MS-DOS file names with absolute or relative paths to be specified within the OMF.
- Facilitates software development on similar or diversified remote hosts.

- Supports file transfers from a remote host using standard file transfer file utilities instead of requiring a specialized tool which changes the file names within the absolute OMF) file.

- Supports the use of Remote File Access (RFA) or Network File System (NFS) to access source and symbol files across a network.

The HP64_DEBUG_PATH Variable.

SRU checks for the environment variable HP64_DEBUG_PATH to determine the location of source and symbol files at run-time. This variable is similar to the "PATH" variable in the shell but is used to find source and symbol files. If this variable is defined, searches will be performed on the path(s) it specifies. Multiple directories are are seperated by a colon (:). For example, in shell notation:

```
HP64_DEBUG_PATH=/sources:/users/me/src:%
export HP_DEBUG_PATH
```

will tell SRU to first look for files in the directory /sources, then in the directory /users/me/src, and finally by the path specified in the absolute (OMF) file (designated by the % character).

When using the HP64_DEBUG_PATH variable to search in a directory, SRU will strip off any directory path information from the original file name before prefixing the search path. File naming conventions are determined by the existence of their respective directory delimiters: slash (/) for UNIX, square or angle bracket pairs ([] or < >) for VMS and backslash (\) for MS-DOS. If the original file name is determined to be a VMS or MS-DOS file name, SRU will also strip off any VMS version number and semicolon delimiter (;), then look for the file using a case-insensitive search.

For example, if SRU needs to access a source file referred to in the absolute (OMF) file as USER$DISK:[BARB]MAIN.C;5, the search will be performed using the file name MAIN.C. Assuming this file was moved to /sources/main.c, SRU should find it correctly.

If the HP64_DEBUG_PATH variable is not defined, SRU will search for files in the following order:

1. Using a file name translation table, (described later in this section).
2. Using the path specified in the absolute file.
3. In the current working directory.

File Name Translation Table

If the HP64_DEBUG_PATH contains a double percent directive (%%) or the default search path is used, SRU will use a file name translation table (mapfile) to locate files. A mapfile is an ASCII file providing a one-to-one mapping of file names appearing within the absolute (OMF) file to corresponding UNIX files accessible on the local file system.

The mapfile should have the same name as the absolute (OMF) file with the extension ".MP". For example, if the name of your absolute file is /myproject/builder.X, the mapfile should be called /myproject/builder.MP. In addition to this mapfile (which is considered to be specific to a particular absolute) SRU will also look for a global mapfile in your login directory. The name of this file is $HOME/.mapfile.MP. If both files exist, both will be used.

The format of the mapfile is simple:

- Each line in the mapfile should contain a from-pattern (the original file name appearing within the absolute (OMF) file), a to-pattern (the alternate file name accessible on the local file system) and an optional comment, all separated by white space (blank(s) or tab(s)).
- White space before the first field will be ignored.
- Comments are delimited by a '#' and a new line.
- Either pattern may be optionally delimited by double quotes.
- A limited form of wildcarding is also supported:
  - A single asterisk may appear at the beginning of from-pattern (a suffix pattern), at the end of

from-pattern (a prefix pattern), or as the entire from-pattern.

- The asterisk matches any string , including the null string.
- The asterisk may appear anywhere in to-pattern.

- If an input file name matches the from-pattern, the file name will be converted to the form defined by the to-pattern.

For example, consider the the following mapfile which contains two exact mapping patterns and three wildcard mapping patterns:

```
HP$DISK:[MYPROJECT]MAIN.C;2     /sources/main.c     #exact match
HP$DISK:[MYPROJECT]FILE1.C;5    /sources/file1.c    #exact match
/bozo/*                         /users/me/*         #prefix pattern
*.C                             *.c                 #suffix pattern
*                               /net/remotesys*     #add prefix
```

Given the file name HP$DISK:[MYPROJECT]FILE.C;5, SRU would map the file name to /sources/file1.c (by applying the exact match rule) before checking to see if the file exists. Similarily, a file named /bozo/glomp.c would be mapped to /users/me/glomp.c by applying the prefix-pattern rule. A file named /users/barb/main.c would be mapped to /net/remotesys/users/barb/init.c by applying the add prefix rule.

Algorithm for Mapping File Names

The following steps are performed to map the input file name using the file name translation table. When testing for matches using a wildcard, if there are multiple prefix- or suffix-patterns that match the input file name, the one matching the more explicit characters is used.

Step 1. Test for an exact match.
Does the input file name match a from-pattern without a wildcard?

Yes	Search ends. The associated to-pattern is the mapped file name to look for.

No	Go to step 2.

Step 2. Test for prefix-pattern match.
Does the input file name match a from-pattern ending in a wildcard?

Yes	Transform the input file name as specified by the associated to-pattern. Characters from the input file name that match the wildcard replace the wildcard in the to-pattern. Go to step 3.
No	Go to step 3.

Step 3. Test for suffix-pattern match.
Does the input file name (or the transformed file name from step 2) match a from-pattern starting in a wildcard?

Yes	Transform the input file name as specified by the associated to-pattern. Characters from the input file name that match the wildcard replace the wildcard in the to-pattern. Go to step 4.
No	Go to step 4.

Step 4. Was the input file name transformed in step 2 and/or step 3?

Yes	Search ends. The transformed file name is the mapped file name to look for.
No	Search ends. The input file name is the mapped file name to look for.

Examples of Use

The following scenarios describe the various development environments and what the user must do to run emulation.

Scenario 1

Software development and debug are performed on the same host; source and symbol files are not moved.

1. The user compiles/links program on local HP 64000-UX host.

2. The user starts an emulation session. The user will see and reference the original file names. Files can be referenced by basename, relative path (must be in the correct directory) or absolute path. No mapping of file names is required to access source and/or symbol files.

Scenario 2

Software development and debug are performed on the same host; source and symbol files are moved to another directory.

1. The user compiles/links program on local HP 64000-UX host.

2. The user moves all source and symbol file(s) to a different directory (ie: /test).

3. The user creates a file name translation table so that source and symbol files will be accessed in test. For example, if the original files were located in the directory /users/me, the file name translation table could be created with the following commands:

   srubuild -ln /test/glomp.X | mapfn **-p**'/users/me/* /test/*' /test/glomp.MP

   The srubuild command will print out a list of file names appearing within the OMF file and the **mapfn** command will translate this list into a list of file name pairs, with the original file name first and the mapped file name second.

   Alternatively, the user defines a directory search path using the environment variable HP64_DEBUG_PATH.

For example, if the executable was originally created in directory /users/me, and all source and symbol files are now located in the subdirectories srcdir1 and srcdir2:

```
HP64_DEBUG_PATH=/test/srcdir1:/test/srcdir2
```

4. The user starts an emulation session. The user will see the original file names of /users/me. The user can reference files with relative or absolute paths under /users/me; files cannot be referenced with absolute paths under /test. The file name translation table and/or the HP64_DEBUG_PATH environment variable will force emulation to access the correct source and symbol files under /test.

Scenario 3

Software development and debug are performed on different hosts; HP 64000 format absolute files are created; source and symbol files are transferred to the debug host (original pathnames may or may not be retained).

1. The user compiles/links program on the remote host.

2. The user moves all source and symbol file(s) from the remote host to the local HP 64000-UX host.

3. If the pathnames are the same in both lists, the user simply starts an emulation session. Files can be referenced by basename, relative path (must be in the correct directory) or absolute path. No mapping of file names is required to access source and/or symbol files.

4. If the original pathnames are not retained, the user creates a file name translation table and/or defines a directory search path as described above.

5. The user starts an emulation session. The user will see the original file names. The user can reference files by original relative or absolute paths; files cannot be referenced using absolute paths on the local system. The file name translation table and/or the HP64_DEBUG_PATH environment variable will force emulation to access the correct source and symbol files on the local system.

Scenario 4

Software development and debug are performed on different Sun hosts; source and symbol files are remotely accessed using RFA (original pathnames are not retained).

1. The user compiles/links program on a remote host.

2. The user logs in to HP 64000-UX host and establishes a netunam connection back to the remote development host.

3. The user creates a file name translation table. For example, if the name of the remote system is **remotesys**, a file name translation table could be created in the user's home directory with a single wildcard pattern:

   ```
   echo "/* /net/remotesys/*" $HOME/.mapfile.MP
   ```

   Alternatively, the user defines a directory search path using the environment variable HP64_DEBUG_PATH. For example, if all source and symbol files are located in the subdirectories srcdir1 and srcdir2:

   ```
   HP64_DEBUG_PATH=/net/remotesys/users/me/srcdir1:
   /net/remotesys/users/me/srcdir2
   ```

4. The user starts an emulation session. The user will see the original file names without /net/remotesys. Files can be referenced by original, relative or absolute paths; files cannot be referenced by absolute paths beginning with /net/remotesys/.... . The file name translation table and/or the HP64_DEBUG_PATH environment variable will force emulation to access the correct source and symbol files under /net/remotesys/users/me on the remote system.

Scenario 5

Software development and debug are performed on different hosts; source and symbol files are remotely accessed using NFS (original pathnames may or may not be retained).

1. The user compiles/links program on a remote host.
2. The user logs in to 64000-UX host and has access to remote files using NFS set up by the system administrator.
3. If the original pathnames are retained, the user simply starts an emulation session. Files can be referenced by basename, relative path (must be in the correct directory) or absolute path. No mapping of file names is required to access source and/or symbol files.
4. If the original pathnames are not retained, the user creates a file name translation table or defines a directory search path as described above.
5. The user starts an emulation session. The user will see the original file names. Files can be referenced by original relative or absolute paths; files cannot be referenced by their alternate pathnames. The file name translation table and/or the HP64_DEBUG_PATH environment variable will force emulation to access the correct source and symbol files on the NFS mounted file system.

Note

NFS products on VAX computers vary. VAX source files will need to be converted to HP ASCII for successful emulation.

Messages Generated By SRU

Error Messages

"Absolute file <file name> is newer than .GY file."

The absolute file has been modified since the SRU database files were built.

What might have gotten you there:

The modification date associated with your absolute file became newer than the dates on the SRU database files.

What to do:

Rebuild the SRU database files.

"ABSTXT not found"

What might have gotten you there:

Loading an OMF386 file that has no absolute data.

What to do:

No absolute data can be loaded into memory since there is none in the file.

"Address error: <explanation>"

Internal software error.

What might have gotten you there:

Users should not see this message. This implies a defect in the database software. Please try to remember and/or reproduce the steps you used to produce the problem.

What to do:

Copy the exact text of the message and contact your nearest HP representative.

"**Ambiguous name: <symbol name>**"

More than one symbol referred to in the absolute file is named <symbol name>.

What might have gotten you there:

Using duplicate symbol names for symbols.

What to do:

Rename the symbols that have duplicate names.

"**Bad name passed to database (<name>)**"

A call was made to the database with incorrect parameter data.

What might have gotten you there:

Users should not see this message. This implies a defect in the database software. Please try to remember and/or reproduce the steps you used to produce the problem.

What to do:

Copy the exact text of the message and contact your nearest HP representative.

"**Bad processor id in <file name>: <processor-id name>**"

A processor id read from the absolute file cannot be handled by the version of SRU database software that you are using.

What might have gotten you there:

Reference to a absolute file that is not the correct format for the processor tool you are trying to use.

What to do:

Reference (or create) a different absolute file.

"**Bad query made to database (<function name>)**"

A call was made to the database with insufficient data or it was made at a time when the database was not set up for the call.

What might have gotten you there:

Users should not see this message. This implies a defect in the database software. Please try to remember and/or reproduce the steps you used to produce the problem.

What to do:

Copy the exact text of the message and contact your nearest HP representative.

"**Builder cannot read <file name>**"

The software needed to read the absolute file <file name> is not part of the SRU database software linked into your product.

What might have gotten you there:

You are using the incorrect HP product to handle your absolute file.

What to do:

Obtain the correct HP product.

"**Cannot build database: <.GY file name>**"

The global symbol information associated with an absolute file cannot be built.

What might have gotten you there:

At the time that you were trying to build a database, there was another process also trying to build the database. This other process failed in its build and your process detected this.

What to do:

Query the other process to see why the build failed or retry the build in your process to find out why it fails.

"**Database <file name> cannot be used**"

The .GY file for <file name> had inconsistent data within it as compared to the information within the absolute file and therefore, the .GY file cannot be used.

What might have gotten you there:

Your absolute file might have been recompiled with a compiler for a different processor than it was compiled for when the .GY database file was generated.

Or the global database (.GY) may have been built for one product with a older version of the data base software than the version that the product you are trying to use has

linked in. This will not usually be a problem, but it is possible for HP tools to exist that create/use incompatible data bases.

On initial release of the database software, all .GY's will be compatible for the same processor file formats. It is possible that future releases of the database software will result in incompatible databases among products and processor file formats.

What to do:

Regenerate the .GY file using the srubuild command (and the -p option) or regenerate it using the consumer tool (e.g. emulation).

"**DEBTXT not found**"

What might have gotten you there:

Loading an OMF386 file that has no debug information.

What to do:

You will have no symbols available until you load a file that has symbols (in a DEBTXT section).

"**<file name>: absolute file is already open**"

<file name> has already been opened by the SRU software (a database may only be opened once per database session).

What might have gotten you there:

Users should not see this message. This implies a defect in the database software. The database software should prevent multiple opens on the same absolute. Please try to remember and/or reproduce the steps you used to produce the problem.

What to do:

Copy the exact text of the message and contact your nearest HP representative.

"**<file name>: <strerror() explanation of problem>**" - An I/O error was detected.

What might have gotten you there:

You were trying to access a file that is not readable/writable or does not exist or has incorrect permissions.

What to do:

Check file names, file permissions, etc.

"**<file name>: unexpected end of file**"

I/O error when reading <file name>.

What might have gotten you there:

Refer to UNIX man pages read(2) and write(2)

What to do:

Check the permissions on your files and their contextual validity. Perhaps regenerate files. Refer to UNIX man pages read(2) and write(2).

"**Global symbols not found**"

What might have gotten you there:

Making a symbol-to-address request when there is no symbol database, e.g. "display memory MY_SYMBOL".

What to do:

Load an absolute that contains symbols or don't try to use symbol names on the command line.

"**Incorrect abs/sym mode when reopening file**"

Internal software error.

What might have gotten you there:

Please try to remember and/or reproduce the steps you used to produce the problem.

What to do:

Contact your nearest HP representative.

"**Incorrect file format in <file name>**"

The reader for an absolute file detected an error in the absolute file during reading.

What might have gotten you there:

An absolute file was incorrectly modified and no longer meets the format specifications for its format.

What to do:

Regenerate the absolute file or refer to a different absolute file.

"**Internal error in <function name>**" - Database software problem.

What might have gotten you there:

Users should never see this error message. This occurs when the software detects an unrecoverable fault condition. We can't predict what might get you there since it should not be possible to get this message. Please try to remember and/or reproduce the steps you used to produce the problem.

What to do:

Copy the exact text of the message and contact your nearest HP representative.

"**MMU Info not found**"

What might have gotten you there:

Loading an OMF386 file with no absolute data or no Global Descriptor Table/Local Descriptor Table (GDT/LDT) information.

What to do:

All addresses will be interpreted as "Real Mode" addresses. If the file was supposed to be a "Protected Mode" file, it is possible that an error was made during the build phase. HP consumers of OMF386 rely on the presence of GDT/LDT information in the bootloadable file so the absolute file must be built with this information.

"**No absolute file: <file name>**"

The absolute file <file name> does not exist.

What might have gotten you there:

Reference to an incorrect absolute file name or to an absolute file that was deleted or to an absolute file with no read permission.

What to do:

Regenerate your absolute file or change its permissions.

"**No absolute file, No database: <file name>**"

Both the absolute file and the .GY SRU database file for <file name> do not exist.

What might have gotten you there:

Use of an incorrect absolute file name.

What to do:

Refer to a different absolute file name or regenerate your absolute file.

"**No database: <file name>**"

The SRU database file <file name> does not exist.

What might have gotten you there:

In an attempt to read information from a previous .GY file, the file was found to not exist.

What to do:

Regenerate the .GY file using the srubuild command or from within the HP product.

"**No symbol file: <file name>**"

The .GY SRU database file is corrupt or a HP 64000 format symbol file (.L or .A file) could not be found.

What might have gotten you there:

The .GY file in your directory is corrupt or a HP 64000 format .L or .A file was deleted.

What to do:

Regenerate the .GY file using the srubuild command or from within the HP product OR regenerate the appropriate .L or .A file.

"**Reader called with wrong request: <function name>**"

Internal database software error.

What might have gotten you there:

Users should not see this message. This implies a defect in the database software. Please try to remember and/or reproduce the steps you used to produce the problem.

What to do:

Copy the exact text of the message and contact your nearest HP representative.

"**Reader error detected in <function name>**"

The database builder detected an error in the data passed back by the file format reader software.

What might have gotten you there:

Users should not see this message. This implies a defect in the database software. Please try to remember and/or reproduce the steps you used to produce the problem.

What to do:

Copy the exact text of the message and contact your nearest HP representative.

"**Segment not found**"

What might have gotten you there:

Asking for the segment of a symbol when that symbol does not contain the segment, e.g. "display memory file.c: segment DATA". If that file does not contain any memory in a segment named "DATA", then this message will be seen.

What to do:

Don't ask for segments that are not associated with the symbol.

"**Source reference not found**"

What might have gotten you there:

Requesting the address of a source line number that does not exist, e.g. "display memory file.c: Line 87".

What to do:

Display symbols in the file to see which line numbers exist, e.g. "display local_symbols_in file.c:". Many language systems do not generate source references for every source line, only source lines that generate code.

"**Subtree <.LY file name> cannot be accessed.**"

The .LY with the local symbol information cannot be built (nor accessed) OR its information is out of date with respect to the absolute file.

What might have gotten you there:

The SRU database files in the .Ys directory may have been deleted or permissions modified so that the database files cannot be correctly accessed. References to symbols that are not present in the absolute file might also generate this error. For example, library modules are often present in the absolute code, but symbols are usually stripped. If you use a library module called "math.c" and enter the command "display local_symbols_in math.c:", you may see this message.

What to do:

You will be unable to refer to symbols in the module.

"**Symbol not found**" - Information for a symbol was not found in the database.

What might have gotten you there:

Symbol was referenced whose information is not contained within the absolute file.

What to do:

Check the symbol name or the valid symbols in your absolute file.

"Transfer address not found"

What might have gotten you there:

> "run from transfer_address" when there is no transfer address. Some file formats (e.g. OMF386) do not contain a transfer address. Most language systems that support the concept of a transfer address require the user to explicitly specify this address.

What to do:

> Check your language tools to see what you must do to ensure that there is a transfer address in the absolute file.

"Unrecognized file type - <file name>"

The file <file name> cannot be handled by the version of SRU database software linked into your product.

What might have gotten you there:

> The absolute file you are loading in is of a format that cannot be identified by the SRU database software.

What to do:

> Use a different absolute file or a different HP tool.

"Write error: exceeded allotted capacity of file <file name>"

The attempt to write to a file exceeded the allotted capacity of the file either because of the current ulimit() or because the physical end of the medium was detected.

What might have gotten you there:

> The current ulimit() is too low or you need a bigger disk.

What to do:

> Reset ulimit(); install a larger disk.

IEEE-695 Specific Error Messages

"File error: An address is not in the range of a BB11"

A BB11 is a part of an IEEE-695 file that describes the address ranges of a module. The BB11 must describe all the address ranges of all the symbols that appear in a module. It is an error for an IEEE-695 file to contain a symbol whose address is not encompased by an address range in the BB11 associated with the module.

What might have gotten you there:

This message indicates a defect in the tool that produced the IEEE-695 file. In most cases fixing this defect will be beyond the users control.

What to do:

Contact your nearest HP representative. If the language tool that produced the IEEE-695 file was not HP, you may also need to contact their representative.

"File error: No BB11 found in module <module name>'s BB10 scope"

A module did not have a BB11 part. The BB11 part describes the address ranges of the module. It is an error for this part of the file to be absent.

What might have gotten you there:

This message indicates a defect in the tool that produced the IEEE-695 file. In most cases fixing this defect will be beyond the users control.

What to do:

Contact your nearest HP representative. If the language tool that produced the IEEE-695 file was not HP, you may also need to contact their representative.

"File error: Type (TY) record has invalid index"

A type record is used to describe a user defined type such as a structure. Each type record contains an index so that data elements can simply refer to the type by its index. Some index numbers are reserved for pre-defined types. It is an error for an IEEE-695 file to contain a type record that uses one of these reserved indexes.

What might have gotten you there:

This message indicates a defect in the tool that produced the IEEE-695 file. In most cases fixing this defect will be beyond the users control.

What to do:

Contact your nearest HP representative. If the language tool that produced the IEEE-695 file was not HP, you may also need to contact their representative.

"Form C LR/LT record in data part"

An LR or LT record can be used to specify data to be loaded into memory. There are three forms of an LR or LT record. Form A consists of an array of data to be loaded in memory. Form B and Form C are used for relocation. A Form B LR or LT record can safely be ignored, but a Form C record in an absolute file indicates a fatal error. It is pointless to continue to attempt to read the data part after encountering a Form C LR or LT record.

What might have gotten you there:

This message indicates a defect in the tool that produced the IEEE-695 file. In most cases fixing this defect will be beyond the users control.

What to do:

Contact your nearest HP representative. If the language tool that produced the IEEE-695 file was not HP, you may also need to contact their representative.

"Module <module name> could not be read"

What might have gotten you there:

The IEEE-695 file was modified between the time it was opened and the time of the access that produced this message.

What to do:

Rebuild the IEEE-695 file and the SRU database, if the error persists contact your nearest HP representative.

Warning Messages

Note

Warnings that are generated by SRU are written to files to be referenced later. Warnings will usually be one line long. However, if a warning spans two lines, it will be followed by a blank line in the file to enhance readability.

"**Adjacent source references in <file> at line <#>**"

If an absolute contains source references that are adjacent (consecutive when sorted by line and column) aand have the same address, they are considered to be the same source reference and their source ranges are merged.

"**Identical source references in <file> at line <#>**"

If an absolute contains more than one source reference for a given line and column in a given module, those source references are considered to be identical. Identical source references are merged, that is; only one source reference will appear in the database but it will have multiple code ranges, one for each occurrence of that source reference in the absolute file.

"**No symbols in <file>**"

The builder attempted to build a subtree, but there were no symbols for that subtree. An empty subtree was created and there will be no further attempt to build this subtree until the entire database is rebuilt (that is; the next execution of srubuild).

"**Symbol <name1> was renamed to <name2> to avoid name conflict**"

Generated when symbols of the same type with the same name are detected in an absolute. They will be renamed by the SRU database software and the warnings generated will tell you the new name of the symbol. You should refer to the symbols using the new name assigned by SRU when using the HP product in order for the SRU database software to be able to derive information about the symbol.

IEEE-695 Specific Warning Messages

"HP/IEEE file offset (ASWx) was null. Part access not possible"

The IEEE-695 file consists of several parts, each with a set of related information. The first part of the file contains an index which specifies the locations of all the other parts. The ASWx record is used to specify the location of each part; if the ASWx record is null, the corresponding part of the file is missing or inaccessable.

What might have gotten you there:

> The IEEE-695 file does not have an ASW record specifying the location of the debug part. Some language tools require the user to use a "debug" option on the command line in order to enable the generation of all the parts of the IEEE-695 file.

What to do:

> Rebuild your program using the "debug" option on all tools involved in translating and linking your code. If that fails, contact your nearest HP representative. If the language tool that produced the IEEE-695 file was not HP, you may also need to contact their representative.

"Section with address <hex address> was renamed <section name>"

The IEEE-695 file contains records for describing memory address regions, called sections, including the name of the sections. These section names are required for the incremental building feature of SRU. If section names are omitted in the IEEE-695 file, a name will be created by SRU.

What might have gotten you there:

This message indicates that two sections in the IEEE-695 file were given the same name. This will most likely occur when absolute sections have been given a name consisting of a single blank i.e. " ".

What to do:

You may use the absolute file and SRU symbol database. The message is intended only to make you aware of the section name that will be used. If you are concerned, contact your nearest HP representative. If the language tool that produced the IEEE-695 file was not HP, you may also need to contact their representative.

"No source line information was found for module name"

The IEEE-695 file consists of several parts, each with a set of related information. The debug part contains symbol information for the program. Each module may have a set of language symbols and a set of source references.

What might have gotten you there:

A module in the IEEE-695 file does not contain source reference information.

What to do:

Review the documentation for the language tools that produced the IEEE-695 file. Some tools may require a special "debug" option to produce source references. Rebuild your program using the "debug" option on all tools involved in translating and linking your code. If that fails, contact your nearest HP representative. If the language tool that produced the IEEE-695 file was not HP, you may also need to contact their representative.

"Form B LR/LT record in data part"

An LR or LT record can be used to specify data to be loaded into memory. There are three forms of an LR or LT record. Form A consists of an array of data to be loaded in memory. Form B and Form C are used for relocation. A Form B LR or LT record can safely be ignored, but a Form C record in an absolute file indicates a fatal error. It is pointless to continue to attempt to read the data part after encountering a Form C LR or LT record.

What might have gotten you there:

> This message indicates a defect in the tool that produced the IEEE-695 file. In most cases fixing this defect will be beyond the users control.

What to do:

> If you are concerned, contact your nearest HP representative. If the language tool that produced the IEEE-695 file was not HP, you may also need to contact their representative.

How To Use The Srubuild Command

Using the srubuild command requires the following syntax:

srubuild [options] filename [buildfile...]

Where:

filename is the path name (full or relative to the current directory) of the program (i.e. absolute file) whose symbolic database is to be built. The files created for the database will be placed in the subdirectory **filename.Ys** located in the same directory as **filename**.

Options are:

-a addresstype srubuild can usually determine the correct address-driver to use just by looking at the absolute file. However, in the event the absolute file does not contain the necessary information, this option can be used to instruct srubuild to use a specific address-driver when building the database. To get a listing of valid address types, enter an illegal value in the command; for example type in the command srubuild -a XXX.

-l causes a list of the source file names whose source files were used to create **filename** to be copied to stdout (standard output file). This option is used to generate a list of source files that may need to be transferred from a remote system that the absolute was linked on, to the system which needs access to the source files (for displaying source-references). By default no list is produced.

-m messagefile specifies the name of the file in which you want to append error and warning messages that may be produced during the building of

the database. If neither the **-m** or **-q** options are specified, the default message file **filename.Ys/bldmessages.WY** will be used.

Note that messages generated during subsequent building of local symbols, by either srubuild or HP 64000-UX products that perform "on demand" building, will continue to use the same message file that was in effect when the global symbols were built.

-p product specifies the product that will be using the database that is built. If this option is not specified, srubuild will build the database in a manner it selects as appropriate for the absolute file.

-q This option may be used instead of **-m** to completely suppress writing out of messages produced during the building of the database. If the **-m** option is be specified, the **-q** option will be ignored.

Note that srubuild and HP 64000-UX products that perform "on demand" building will continue to suppress build messages during all subsequent local symbol builds until the global symbols are rebuilt again.

-v prints the results of each of the global and local builds to the stderr (standard error file).

By default, the entire database is built. This means that data on all source files will be created. If you are only interested in a subset of the source files, however, you can reduce the time and disk space required by using the following options:

-g Only global symbols will be built; this means that no local symbols will be built at this time. However, local symbols will be

available because the local symbols will be built "on demand" when necessary.

This means that you can determine what subset of source files you are interested in when you are in an HP 64000-UX program instead of deciding ahead of time.

This option is ignored if either the **-f** option is used or one or more buildfile arguments are given.

-f buildfilelist — allows you to specify a subset of the program modules or source files to be processed. This means that the database for the local symbols defined by those program modules or source files is created by this command; when the local symbols are used in an HP 64000-UX program the data does not have to be built.

The **buildfilelist** consists of a number of lines. On each line is the name of a program module or the absolute path of a source file. Blank and comment lines beginning with the '#' character will be ignored.

-n — requests srubuild to only scan the absolute file without building the database. This option is only useful in conjunction with the **-l** option to get a list of source file names from an absolute file without incurring the overhead of building the database.

Examples Using Srubuild

A convenient place to invoke srubuild is within a makefile. Part of an example makefile includes:

```
CC = cc68000
prog : program.o file1.o file2.o
  $(CC) -o prog program.0 file1.o file2.o
  srubuild prog
```

Sophisticated makefiles can be used to create the file list (for the **-f** flag). The following rule (used to make a relocatable file from a C source file) appends the assembly symbol file name to a file named "interesting", for example, only when the file is updated. This "interesting" file (file list) can then be used with the **-f** flag to specify "interesting" files. When this method is used, it is necessary to truncate this file on occasion since the "interesting" files are likely to change over time. For example:

```
CC = cc68000
prog : program.o file1.o file2.o
  $(CC) -o prog program.o file1.o file2.o
  srubuild -f interesting prog
```

Using Sruclean

What Is In This Chapter?

The information in this chapter includes:

- What Sruclean Is.
- How To Use The Sruclean Command.
- Examples Using Sruclean.

What Is Sruclean?

HP 64000-UX products make use of the SRU database to get access to symbolic information produced by HP 64000-UX (or compatible) language systems (compilers, assemblers, and linkers). The database is placed in a subdirectory, which has the name of the absolute with ".Ys" appended.

The sruclean utility will remove or clean up the database that is created for each absolute file (created either by the srubuild command or by the HP 64000-UX products).

The directory or file arguments are interpreted as follows:

- If the argument ends in ".Ys" and is a directory, only the files in that directory will be modified as specified by the options.
- If the argument is a directory that does not end in ".Ys", the disk will be searched starting with that directory, and all files in subdirectories that end in .Ys will be modified as specified by the options.
- If the argument is not a directory, it is assumed to be an absolute file, and the SRU files associated with that absolute file will be modified as specified by the options.

You must specify at least one directory or file. If you want your entire file system to be searched (including network mounted file systems) use:

sruclean <options> /

How To Use The Sruclean Command

Using the sruclean command requires the following syntax:

sruclean [**-v**] [**-m**] [**-r**] [**-o**] [directory ...] [absolute file ...]

where:

-v	Prints actions (to stderr) as they happen. This will also print out how many bytes were deleted from the disk (if the **-r** option was used), or how many lines removed from the "bldmessages.WY" file (if the **-m** option was used).
-m	Cleans up the "bldmessages.WY" file. During srubuild or incremental builds, information about the build is appended to this file. Unless this option is used, the file will grow without bounds. So, you should consider using this in a crontab.
-r	Removes all SRU files and the ".Ys" directory. You could also use the rm -fr <sru directory> command, but -r is somewhat safer in that it guarantees that only directories ending in .Ys will be removed.
-o	This will remove any obsolete SRU files. Usually obsolete files are removed automatically when srubuild is run. Under certain conditions, however, they will have to be removed with this option.

If no options are given, it is equivalent to the command:

sruclean -m -o

Examples Using Sruclean

Example 1: **sruclean -r control race**

This command will remove all SRU files associated with the absolute files **control** and **race**.

Example 2: **sruclean /users**

This command will remove all obsolete SRU files and clean up message files under the directory **/users**.

Example 3: **sruclean -m *.Ys**

This command will clean up the message file in all SRU directories in the current directory.

Example 4: **0 3 * * 6 usr/hp64000/bin/sruclean /**

This **crontab** entry will remove all obsolete SRU files and clean up message files every week starting at 3:00 AM on Saturday.

Using Sruprint

What Is In This Chapter?

The information in this chapter includes:

- What Sruprint Is.
- How To Use The Sruprint Command.
- Examples Using Sruprint.

What Is Sruprint?

Sruprint is a utility that converts the contents of the SRU symbolic database file to a readable format and causes the contents of the file to be printed to stdout (normally your terminal unless it is redirected to a file).

The sruprint utility will only print symbols that are already in the database; it will not build symbols for modules that have not been built.

How To Use The Sruprint Command

Using the sruprint command requires the following syntax:

sruprint [options] absolutefile [<symbol> ...]

Where:

absolutefile is the absolute file built by sruprint. If <symbol> does not follow absolutefile, the global symbols in the absolute file will be printed.

If one or more **<symbol>** arguments follow the absolutefile, they are assumed to be symbol names. This specifies that you also want symbols that are their children to be printed.

Options are:

-f symbolfile allows you to specify a list of symbols in a file rather than on the command line.

The **symbolfile** consists of a number of lines. On each line is the name of a symbol. Blank and comment lines beginning with '#' character will be ignored.

-g suppresses printing of the global symbols.

Examples Using Sruprint

Example 1:

sruprint myexec.X

This command only prints the global symbols associated with the absolute file **myexec.X**. Global symbols include global procedure names, global variables, and file names. When this command is used, the listing will include the symbol name, logical address, and the symbol's offset from the start of the segment which contains the symbol.

Example 2:

sruprint -f my_sources myexec.X

This command prints the global symbols and children of the sources files listed in **my_sources** command file.

my_sources looks like:

/users/me/controller/main.c:
/users/me/controller/sighandle.c:R/users/me/controller/except.c:
/users/projA/library/except.c:
/users/projA/library/margin.c:

Example 3:

sruprint -g -f my_sources myexec.X

This command prints only the symbols that are children of the source files listed in **my_sources**. See example 2 for what the **my_sources** command file looks like.

Using Sruaccess

What Is In This Chapter?

The information in this chapter includes:

- What Sruaccess Is.
- How To Use The Sruaccess Command.

What is Sruaccess?

Sruaccess is a utility that allows interactive examination of virtually all data within the database.

Note

Sruaccess is an unsupported utility program. It is not part of any Hewlett-Packard product and is provided at no cost. Hewlett-Packard makes no warranty on its quality or fitness for a particular purpose. However, for your convenience, its purpose and use are described in this chapter.

How To Use The Sruaccess Command

Using the sruaccess command requires the following syntax:

sruaccess [-a <addrtype>] [-f <fmt-options>] [-p <product>] [abs-filename]

Where command line options are:

-a <addrtype>	Sets the default address type to **addrtype**; see the command **setaddrtype** for more information.
-f <fmt-options>	Sets the default print format to **<fmt-options>**; see the **fmt** command for more information.
-p <product>	Use the builder process that is shipped with **<product>**. If this option is not specified, a builder will be selected from available builders on the system when an **open** command is executed. See the command **prodid**.

Sruaccess Commands

The following commands are accepted by sruaccess:

absrec [startflag]	If **startflag** is not present or if it is "F", print the next absolute record. If **startflag** is "T", get the first absolute record.
childinfo	Give info about the current working symbol's children.
close	Closes the current database.

cd <name> and **cs <name>**

Change the current working symbol to the symbol named **<name>**.

create <name> Create a database for the absolute file **<name>**.

fmt [options] Defines the format that symbols are printed in. Use the command with no options to get a list of currently-supported formats.

fseg Print fsegment information for the current symbol. An fsegment is a mechanism used to implement incremental builds. This command is useful to the SRU implementers and probably to no one else.

lc [-r] [-p <pattern>] [<type>] and
ls [-r] [-p <pattern>] [<type>]

Print the current working symbol's children, using the format last defined with the **fmt** command. The **<type>** of the children to be listed can be specified.

The **-r** option lists the children recursively (normally only the symbol's immediate children are printed).

The **-p <pattern>** command requests that all children whose name matches **<pattern>** (an sh(1) style wildcard string) should be printed.

open <abs-filename> [symbolic]

Attempts to open the database associated with the given absolute file. If a database is already open, this will close the previous one.

symbolic (which is either T or F) sets the value of the symbolic parameter to sruopen to either TRUE or FALSE. If not present on the command line, it is set to TRUE.

procid Print the processor id associated with the current database.

prodid <name> Set the product id to **<name>**.

pwd Print the name of the present working directory.

pws Print the name of the current working symbol.

range <start> <end> [<incr> [<type>]]

Print all symbols that are within the specified address range. This is essentially:

```
for (addr = start; addr <= end; addr+= incr)
{
    sym addr [<type>]
}
```

reopen Reopen an already open database. This command is useful to the SRU implementers and probably no one else.

sa Print the attributes associated with the current working symbol. These include the symbol type, name, address range, etc.

setaddrtype [<addrtype>]

This command affects how address are both printed and entered. Normally, addresses are up to 32-bits of physical address - that is; the libaddrII ADDRATPHYSICAL type. When this command is given, all subsequent input

is in the ADDRAT.... type you entered. If the specified addrtype has more than one libaddrII element, they will be printed out separated by colons.

When the command is given, a line will be printed telling what each of the colon-separated elements will be.

The command with no **<addrtype>** will print a list of currently implemented addrtypes.

sourceref <address>

Print the source reference for the specified address.

sym <address> [<type>]

Print the symbol of **<type>** that most closely encloses **<address>**.

xfraddr Get the transfer address.

Data-entry Formats

<address> See the command **setaddrtype**.

<name> This is the name of a symbol. "|" means the root symbol. If a name does not begin with "|", the specification is appended to the current working symbol to form the full symbol name. Other than that, the format of **<name>** is:

<ascii>[-<type>][|<ascii>[-<type>] ...]

Where **<ascii>** is the ascii portion of the SRUNAME, and **<type>**, if present, is the

type portion of the SRUNAME. See below for what **<type>** looks like.

<type> **<type>** is one of the following strings:

high, low, fsegment, static, sourceref, procedure, filename, module, task, special, alltype. If **<type>** is not specified, it will default to **alltype**.

Interactive Versus Batch Mode

The sruaccess utility can be operated interactively or in batch mode. To use batch mode, simply create a file with the commands of your choice and use the shell to direct the file into sruaccess, for example:

```
sruaccess < mycommands
```

Operation of the utility is slightly different in interactive and batch modes. A command line prompt is printed only in the interactive mode. In the interactive mode, commands may be abbreviated to any unique prefix. In batch mode all commands must be completely specified. The interactive mode will return to the command prompt on input of control-C. Both modes will exit on end of file.

Processor ID Names

Refer to the manual page given in appendix D for the list of processor ID names.

Address Types

Refer to the manual page given in appendix D for the list of supported address types.

Finding More Information

The Symbolic Retrieval Utilities software includes on-line "man" page information on the various related commands and files. This information is accessed by using the UNIX **man** command.

Because the "man" page files are installed under the $HP64000 directory, you must first modify the MANPATH environment variable. When using "sh" or "ksh":

```
$ MANPATH=$MANPATH:$HP64000/man:\
> $HP64000/contrib/man; export MANPATH
```

Or, when using "csh":

```
$ setenv MANPATH $MANPATH:$HP64000/man:\
> $HP64000/contrib/man
```

Once the MANPATH environement variable is set, you can access the on-line "man" page information. For example:

```
$ man srubuild <RETURN>
```

On-line "man" pages are included for the following commands:

sruaccess
srubuild
sruclean
sruprint
sruxlate

Index

A

AxDB and Emulators With SRU **12**

E

EDB/SRU compatibility **12**
error messages
  IEEE-695 specific **52**
  SRU **42**

H

-h option **12**
HP-AxLS and SRU **12**

M

man pages **79 - 101**
messages
  error, IEEE-695 specific **52**
  SRU **42 - 57**
  warning, SRU **54**

O

option, -h **12**

P

procspecial **16**
procspecial symbol
  DATARANGE **17**
  ENTRY **16**
  ENTRY0.. **16**
  EXIT **16**
  EXIT0.. **16**
  TEXTRANGE **16**

S

SRU
  --EXPR--, values referred to **25**
  arrangement of symbols **13**
  entering symbols using --EXPR-- **24**
  file mapping **33 - 41**
  file mapping, advantages **33**
  file name mapping, examples of use **38**
  file name mapping, HP64_DEBUG_PATH Variable **34**
  file name mapping, using file name translation table **35**
  file names **30 - 32**
  file names, search algorithms for locating file name symbols **31**

SRU (continued)
full path of a symbol **14**
highlevel/lowlevel symbols, selecting **29**
HP64KSYMBPATH and cws **27**
language dependencies **15**
language sections versus segments **20**
maximum symbol length **24**
non UNIX file names **31**
printing of symbols in trace lists **28**
procedure special symbols **16, 28**
renaming of symbols **24**
segment symbols **20, 28**
segment symbols, linker's view **22**
segment symbols, physical view **22**
segment symbols, symbol tree **21**
symbol attributes **17**
symbol levels, highlevel **19**
symbol levels, lowlevel **19**
symbol types **18**
symbol types, filename **18**
symbol types, fsegment **19**
symbol types, module **18**
symbol types, procedure **18**
symbol types, procspecial **19**
symbol types, static **18**
symbol types, task **19**
symbols in **13 - 29**
UNIX file names **30**
unnamed block symbols **23**
SRU and HP-AxLS **12**
SRU error messages **42**
SRU messages **42 - 57**
SRU warning messages **54**
SRU/EDB compatibility **12**
sruaccess
address types **77**
data entry formats **76**
finding more information **79**
interactive versus batch mode **77**
processor id names **77**
what it is **72**

sruaccess command **71 - 78**
how to use **73 - 77**
options **73**
syntax **73**
sruaccess commands **73**
srubuild command **9 - 61**
examples using **61**
how to use **58 - 60**
options **58**
syntax **58**
srubuild utility, what it does **10**
srubuild, finding more information **79**
srubuild, incremental builds **10**
srubuild, partial builds **11**
srubuild, what it is **10 - 11**
sruclean command **63 - 66**
examples using **66**
how to use **65**
options **65**
syntax **65**
sruclean, finding more information **79**
sruclean, what it is **64**
sruprint command **67 - 70**
examples using **69**
finding more information **79**
how to use **68**
options **68**
sruprint, what it is **68**

U
using the sruaccess command **71 - 78**
using the srubuild command **9 - 61**
using the sruclean command **63 - 66**
using the sruprint command **67 - 70**

W
warning messages, SRU **54**

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse,

operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Hewlett-Packard
Printed in the USA

HEWLETT PACKARD

User's Guide

Absolute File Translator

HP 64700 Operating Environment

Absolute File Translator

User's Guide

HP Part No. B1471-97008
Printed in U.S.A.
March 1992

Edition 2

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

© Copyright 1991, 1992, Hewlett-Packard Company.

This document contains proprietary information, which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

Intellec is a registered trademark of Intel Corporation.

UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and in other countries.

Hewlett-Packard
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901-2197, U.S.A.

RESTRICTED RIGHTS LEGEND Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in subparagraph (C) (1) (ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013. Hewlett-Packard Company, 3000 Hanover Street, Palo Alto, CA 94304. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Printing History

New editions are complete revisions of the manual. The date on the title page changes only when a new edition is published.

A software code may be printed before the date: this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual revisions.

Edition 1	B1471-97005, July 1991
Edition 2	**B1471-97008, March 1992**

In This Book

This book describes the **xlate** aboslute file translator utility. Included are description of the command line options and possible error messages. The book contains the following appendices describing file formats.

Appendix A **I_hex Absolute File Format**. This appendix describes the Intel Hexadecimal Intellec 8/MDS (I_hex) file format.

Appendix B **Intel 286_OMF File Format**. This appendix describes the Intel 80286 bootloadable object module file format.

Appendix C **M_hex Absolute File Format**. This appendix describes the Motorola hexadecimal S-Record (M_hex) file format.

Appendix D **T_hex Absolute File Format**. This appendix describes the Tektronix hexadecimal (T_hex) file format.

Appendix E **Extended T_hex Absolute File Format**. This appendix describes the extended Tektronix hexadecimal (T_hex) file format.

Appendix F **Hewlett-Packard Absolute File Format**. This appendix describes the HP absolute file format.

Contents

1 **Absolute File Translator**

Introduction 9
Xlate Command Syntax 10
xlate(1) 11
Absolute File Translator 11
Synopsis 11
Description 11
Parameters 11
-t<type> 11
input_file 12
Options 12
-f 12
-o<output_file> 12
-d<data width> 12
-b<bus width> 12
-s 13
Examples 13
Error Messages 14
Informational Messages 14
Abort record encountered 14
Bad checksum value, data may be incorrect 14
No end record encountered 14
Fatal Errors 15
Attempt to find current file position was unsuccessful . . . 15
Attempt to move to new file position failed 15
Attempted to read end of file 15
Bad or missing translation file type specification 15
Could not complete translation from HP 64000 format . . 15
Could not complete translation to HP 64000 format 15
Data width base must be 8 or 16 15
Input absolute file name exceeds 253 characters 15
Invalid address, address too large for processor 16
I/O error on HP 64000 absolute file during close 16
I/O error on HP 64000 absolute file during create 16
I/O error on HP 64000 absolute file during open 16

I/O error on HP 64000 absolute file during read 16
I/O error on HP 64000 absolute file during rewrite 16
I/O error on HP 64000 absolute file during write 16
I/O error on HP 64000 binary file during close 16
I/O error on HP 64000 binary file during create 17
I/O error on HP 64000 binary file during open 17
I/O error on HP 64000 binary file during read 17
I/O error on HP 64000 binary file during rewrite 17
I/O error on HP 64000 binary file during write 17
Load file not in correct format for processor 17
Non hex characters in inappropriate location of file 17
Translation not supported by this utility 18
Unable to close absolute file 18
Unable to close the non-HP file 18
Unable to create absolute file 18
Unable to open the non-HP file 18
Unable to perform a file read 18
Unable to perform a write to a file 18
Unable to read absolute file 18
Unable to read record 19
Unable to read XLATE load file 19
Unable to rewind file 19
Unable to translate record 19
Unable to write to absolute file 19
Word count exceeds 125 16 bit words or 250 bytes 19

A I_hex Absolute File Format

Introduction 21
Start Record Example 23
Extended Address Record Example 23
Data Record Example 24

B Intel 286_OMF File Format

Introduction 25
Bootloadable Module Object Files 25
Bootloadable Module Header 26
Partition 26
Table of Contents 27
Absolute Text Section 27
Debug Text Section 28

C M_hex Absolute File Format

Introduction 29
S0 Type 29
S1 Type 30
S2 Type 31
S3 Type 31
S5 Type 32
S7 Type 33
S8 Type 34
S9 Type 35

D T_hex Absolute File Format

Introduction 37
Data Record 38
Example of a Data Record: 38
Terminating Record 39
Abort Record 40

E Extended T_hex Absolute File Format

Introduction 41
Variable-Length Fields 41
Valid Characters 41
Data Record 43
Header Field 43
Symbol Record 44
Section Definition Field 44
Symbol Field 45
Termination Record 46

F Hewlett-Packard Absolute File Format

Introduction 47
Processor Information Record 49
Data Record 50

Absolute File Translator

Introduction

The absolute file translator (*xlate*) is a utility that converts Intel, Motorola, and Tektronix absolute files into HP absolute files, and vice-versa.

You can convert Intel, Motorola, or Tektronix absolute files into HP absolute files, load the converted files, and execute them in the HP 64000 emulation environments.

You also can convert HP absolute files into Intel, Motorola, or Tektronix absolute files. This may come in handy, for instance, when you wish to program ROMs with code developed in the HP 64000 or environments with a prom programmer which does not support HP absolute files, but does support Intel, Motorola, or Tektronix absolute files.

The specific absolute file formats that can be converted into HP absolute files are:

- Intel Hexadecimal Intellec 8/MDS (I_hex).
- Intel 286_OMF (bootloadable modules only).
- Motorola Hexadecimal (M_hex).
- Tektronix Hexadecimal (T_hex).
- Extended Tektronix Hexadecimal.

Note

HP absolute files can be converted into all file formats above **except** Intel 286_OMF files.

Note

The *xlate* utility will **not** convert symbol records.

Xlate Command Syntax

The following pages describe the *xlate* command syntax in the UNIX operating environment. Descriptions of the error messages issued by *xlate* appear at the end of the chapter.

Table 1. Bus and Data Width of Microprocessors

Microprocessor	Data Width	Bus Width
8080/85, Z80	8 bits	8 bits
6800/01/03/05/09	8 bits	8 bits
68HC11	8 bits	8 bits
650X	8 bits	8 bits
68000/08/10	16 bits	8 bits
8021/22, 8041/48	8 bits	8 bits
9900/40/85/89/99	16 bits	8 bits
9980	8 bits	8 bits
99XXX	16 bits	16 bits
1802	8 bits	8 bits
F8	8 bits	8 bits
Z8	8 bits	8 bits
8086/88/89/186/188/286	16 bits	8 bits
Z8001/2	16 bits	8 bits
8051	8 bits	8 bits
1750A	16 bits	16 bits
TMS320	16 bits	16 bits
8096	16 bits	8 bits

xlate(1)

Absolute File Translator

Synopsis

```
xlate -t<type> [-f] [-d<data width>]
[-b<bus width>]  [-o<output_file>]  input_file . .
```

Description

The **xlate** utility is used to convert absolute load files from any of several formats to the HP 64000 absolute file format or to convert HP 64000 absolute files to any of the other file formats. These formats include Intel Hexadecimal, Tektronix Hexadecimal, Extended Tektronix Hexadecimal, Motorola S Record, and Intel 286_OMF.

Note

The **xlate** utility does not convert HP 64000 absolute files into 286_OMF files.

The legal ASCII data in any hex file can only be separated by spaces, tabs, or newlines; otherwise, translation will stop and an appropriate error message will be printed.

Parameters

Definitions for syntactical terms are as follows:

-t<type>

Specifies the type of file to be converted where the valid types are:

ih80	I_hex format (8080/8085).
ih86	I_hex format (8086/8088).
iomf	Intel 286_OMF format.
tek	T_hex format.
etek	Extended T_hex format.

mot Motorola S Record format.

input_file

The name of the file to be converted (any valid UNIX file specification). The output file has the same name with a ".X" extension if it is an HP absolute file, or a ".H" extension otherwise. Specifying multiple input files is the same as entering the **xlate** command for each of the files.

Options

The **xlate** command line options are defined as follows:

-f

This option specifies that the translation will be from the HP absolute file format.

If the **-f** option is not used, the translation will be to an HP 64000 absolute file by default.

-o<output_file>

This option allows you to specify an output file name. By default, the output file has the same base name as the input file with a ".X" appended for HP 64000 format files and a ".H" appended otherwise.

-d<data width>

Data Width Base: This is the size of the smallest addressable unit of the target microprocessor. Legal values are 8 and 16. The default is 8 if this option is not used.

-b<bus width>

Data Bus Width: This is the data bus width of the target microprocessor. Legal values are 8 and 16. The default is 8 if this option is not used.

-s

Scale Address. This option is for use with processors with a 16-bit basic addressable unit (data width base). Normally, when addresses for these processors are encountered in a hex file, they are treated as word addresses; that is, the address is incremented once for each two bytes of data in the file. This is not a completely standard practice, and some vendors treat the addresses as byte addresses. Under these circumstances, the **-s** option must be used to read or produce files that are compatible with other software.

Examples

Here are two examples of the **xlate** command:

```
xlate -ttek -b16 tek_abs_file.H <RETURN>
```

The command above converts a Tektronix Hexadecimal Format absolute file (for a 16-bit microprocessor) into an HP 64000 absolute file.

```
xlate -ftmot -b16 hp_abs_file.X <RETURN>
```

The command above converts an HP 64000 absolute file (for a 16-bit microprocessor with a 16-bit data width) into a Motorola S Record format file.

Note

The file format converter is contained in public directory */usr/hp64000/bin*. If */usr/hp64000/bin* is in the user's directory path, a conversion can be run by using only the command *xlate*.

Error Messages

This section lists, in alphabetical order, the error messages which can occur when using the **xlate** utility. There are two types of messages:

- Informational Messages. The **xlate** utility will issue these messages to warn you about certain situations. The file format conversion will continue.
- Fatal Errors. These errors will cause file format conversion to be halted.

The error messages, along with descriptions of why the error occurred, are listed below.

Informational Messages

Abort record encountered

This message is informational and indicates that an abort record was read at some point during the processing of the input file. The translator will continue translation, and the output file should be usable.

Bad checksum value, data may be incorrect

This message indicates that an incorrect checksum value was read from the file. The translator will print this message once for each record with an incorrect checksum. The translation does not stop, and the output file will be usable provided that the data read from the file was correct despite the incorrect checksum.

No end record encountered

This message indicates that no end record was read from a file format that supports end records. This will not affect translation, except that any information contained in the end record, usually a transfer address, will not be available.

Fatal Errors

Attempt to find current file position was unsuccessful

An attempt to locate the current position in output file failed.

Attempt to move to new file position failed

An attempt to reposition in the output file was unsuccessful.

Attempted to read end of file

The translator encountered end of file while not on a record boundary.

Bad or missing translation file type specification

The filetype specification was omitted or incorrectly specified.

Could not complete translation from HP 64000 format

The translator was unable to translate one of the records in the input file. Other messages will be combined with this message to better indicate the cause of the failure.

Could not complete translation to HP 64000 format

The translator was unable to translate one of the records in the input file. Other messages will be combined with this message to better indicate the cause of the failure.

Data width base must be 8 or 16

The data width base specification is incorrect. Check that the specification is either 8 or 16.

Input absolute file name exceeds 253 characters

The input file name specification is too large. If the specification is correct then you must rename the file to allow the translator to translate the file.

Invalid address, address too large for processor

An attempt was made to create an address (load or transfer) that was to large for the output file format to accommodate.

I/O error on HP 64000 absolute file during close

The HP 64000 format file could not be closed.

I/O error on HP 64000 absolute file during create

The HP 64000 output file could not be created. Check that the file protections allow you to create and write to the output file.

I/O error on HP 64000 absolute file during open

The HP 64000 input file could not be opened. Check that the specified file exists and that the file protections allow you read access to the file.

I/O error on HP 64000 absolute file during read

The HP 64000 input file could not be read. Check that the file protections allow you to read the file.

I/O error on HP 64000 absolute file during rewrite

An attempt to modify the processor information record in the HP 64000 file failed.

I/O error on HP 64000 absolute file during write

The HP 64000 output file could not be written to. Check that the file protections allow you to write to the output file.

I/O error on HP 64000 binary file during close

The HP 64000 format file could not be closed.

I/O error on HP 64000 binary file during create

The HP 64000 output file could not be created. Check that the file protections allow you to create and write to the output file.

I/O error on HP 64000 binary file during open

The HP 64000 input file could not be opened. Check that the specified file exists and that the file protections allow you read access to the file.

I/O error on HP 64000 binary file during read

The HP 64000 input file could not be read. Check that the file protections allow you to read the file.

I/O error on HP 64000 binary file during rewrite

An attempt to modify the processor information record in the HP 64000 file failed.

I/O error on HP 64000 binary file during write

The HP 64000 output file could not be written to. Check that the file protections allow you to write to the output file.

Load file not in correct format for processor

The translator read invalid characters or characters in the wrong location for the specified file type. Check that the specified file type matches the type of file being translated and that the direction of translation is properly specified.

Non hex characters in inappropriate location of file

The translator encountered invalid characters when reading a hex file. This message usually indicates that an attempt was made to translate a file that was not a hex absolute file of the appropriate type.

Translation not supported by this utility

The translator could not perform the requested translation. Check that the file format type is correctly specified. Also, note that translation from HP 64000 format files to Intel 80286 OMF files is not supported.

Unable to close absolute file

The translator could not close one of the files used in the translation.

Unable to close the non-HP file

The translator could not close one of the files used in the translation.

Unable to create absolute file

The translator could not create the specified file. Check that file protections allow the creation of the output file.

Unable to open the non-HP file

The translator could not open one of the files required for the translation. You should check that the input file exists as specified to the translator and that you have the ability to create the output file (i.e., the correct permissions).

Unable to perform a file read

The translator could not read the specified file.

Unable to perform a write to a file

The translator could not write to the output file. Check that file protections allow you to write to the output file.

Unable to read absolute file

The translator could not read the specified file. Check that the file protections allow you to read the input file.

Unable to read record

The translator could not read one of the records in the input file. If this message is accompanied by a message indicating that the input file could not be read, that is the cause of the error; otherwise, there is some problem with the record structure of one of the records in the file.

Unable to read XLATE load file

The translator could not read the specified file. Check that the file protections allow you to read the input file.

Unable to rewind file

An attempt to rewind the input file failed.

Unable to translate record

The translator was unable to translate one of the records in the input file. Other messages will be combined with this message to better indicate the cause of the failure.

Unable to write to absolute file

The translator could not write to the output file. Check that file protections allow you to write to the output file.

Word count exceeds 125 16 bit words or 250 bytes

This is an internal error in the translator contact HP.

I_hex Absolute File Format

Introduction

Intel Hexadecimal Intellec 8/MDS (I_hex) File Format for paper tape records consists of several fixed fields. An example follows:

```
: ll aaaa tt dddd..dd xx
```

(spaces are inserted for clarity, each letter represents one hex character)

Where:

:	record header character
ll	record length (number of data bytes)
aaaa	16-bit load address (or offset for 8086)
tt	record type number (see table A-1)
dddd..dd	hex data (if any)
xx	checksum (xx = 2's complement (ll+aa+aa+tt+dd+dd+...+dd)).

The various Intel record types and their field contents are shown in table A-1.

The older 8080 type transfers consisted of record types 00 and 01 only (see table A-1). The 8086 and 8088 transfers use all four types of records.

Examples of different types of records are shown below.

Table A-1. Intel Record Types

Data Record:

Record type = 00
Record length = nn (two hex digits) # data bytes
Load address = load address
Data = N data bytes

End Record: (:00 0000 01 FF)

Record type = 01
Record length = 00
Load address = *transfer address
Data = none

Extended Address Record: (used with 8086 only)

Record type = 02
Record length = 02
Load address = 00 (meaningless)
Data = USBA (16-bit upper segment base address)

Start Record: (only output for 16-bit processors but is recognized by terminal)

Record type = 03
Record length = 04
Load address = 00 (meaningless)
Data = CS and IP of 8086 processor (Start ADDR)

*NOTE: This field contains the transfer address only when 8080-type format is specified. This field is meaningless when using the 8086/8088 format.

Start Record Example

```
:04 0000 03 0000 1234 B3
```

(spaces added for clarity)

Where:		
	04	byte count of data bytes
	0000	load address (meaningless)
	03	record type = start record
	0000	8086/8088 CS register value
	1234	8086/8088 IP register value
	B3	Checksum = two's complement of: 04+00+00+03+00+00+12+34.

Extended Address Record Example

```
:02 0000 02 1000 EC
```

(spaces added for clarity)

Where:		
	02	number of data bytes
	0000	load address (meaningless)
	02	record type = extended address record
	1000	USBA value. SBA = USBA*10H = 1 0000H
	EC	Checksum = two's complement of: 02+00+00+02+10+00.

Data Record Example

```
:03 0A00 00 E# 7C 47 4D
```

(spaces added for clarity)

Where:	03	byte count
	0A00	16-bit load address
	00	record type = data record
	E3,7C,47	data
	EC	Checksum = two's complement of: 03+0A+00+E3+7C+47.

Intel 286_OMF File Format

Introduction

The **xlate** utility will convert Intel 286_OMF bootloadable module object files into HP absolute format files. **However, the "xlate" utility does not convert HP absolute files into Intel 286_OMF files.** The file format for Intel 286_OMF bootloadable module object files is described below. (The number of bytes in a field, if constant, is shown in parentheses following the field descriptions.)

Bootloadable Module Object Files

A2	The file header for bootloadable files.
Checksum	Equal to the one's complement of the (8-bit) sum of all bytes in the file.

Bootloadable Module Header

Total Space (4)	Date (8)	Time (8)	Creator (41)	GDT Limit	GDT Base (4)	
	IDT Linit (4)	IDT Base (4)	TSS Selector (2)			

Total Space	The total number of bytes of the bootloadable file.
Date	Date of module creation in the form: MM/DD/YY.
Time	Time (military) of module creation in the form: HH:MM:SS.
Creator	Name of the program which created the module.

Partition

Table of Contents	Absolute Text Section	Debug Text Section

Section	Either an absolute text section or a debug text section. Both are described below.

Table of Contents

Abs. Text Loc. (4)	Debug Text Loc. (4)	Last Loc. (4)	Next Partition	Reserved (4)

Abs. Text Loc.	The byte offset of the absolute text section from the start of the file.
Debug Text Loc.	The byte offset of the debug text section from the start of the file (zero if there is no debug text sextion).
Last Location	The location of the first byte after the partition.
Next Partition	The location of the next partition's table of contents, or zero if there is no following partition.
Reserved	These bytes should be zero.

Absolute Text Section

Read Address (3)	Length (2)	Text	. . .	Real Address (3)	Length (2)	Text

Real Address	The load address of the text.
Length	The number of following text bytes.
Text	The absolute bytes to be loaded.

Debug Text Section

No. of Segments (2)	Length (2)	Location (2)	. . .	Location (2)	Text

No. of Segments	The number of debug segments in the file.
Length	The number of location bytes which follow.
Location	The byte offset of a debug segment from the start of the file. The location equals 0 if the debug segment does not exist.
Text	Contains a concatenation of the bytes from the debug segments in the order that their locations are previously listed.

M_hex Absolute File Format

Introduction

Eight Motorola hexadecimal (M_hex) record types are supported. They are distinguished by the eight record headers S0, S1, S2, S3, S5, S7, S8, and S9. The record types are discussed in the paragraphs below.

S0 Type

The S0 type is the start record and may contain miscellaneous information. The format for an S0 type record is as follows:

```
S0 bb yyyyyyyy xx
```

(spaces are inserted for clarity, each letter = one hex character)

Where:		
	bb	is the byte count
	yy...yy	is arbitrary hex data
	xx	is the checksum (see S1 record checksum).

S1 Type

The S1 type is a data record beginning on a 16-bit load address. The format for an S1 type record is as follows:

S1 bb aaaa dd...dd xx

(spaces are inserted for clarity, each letter = one hex character)

Where:

Where:		
	bb	# of data bytes + # of address bytes + 1 (checksum)
	aaaa	16-bit load address
	dd...dd	Data bytes (or words) up to 24 bytes per record
	xx	Checksum = one's complement of: bb+aa+aa+dd+...+dd.

Example: **S1 05 A000 6C 01 ED**

(spaces added for clarity)

S2 Type

The S2 type is a data record starting on a 24-bit load address. The S2 type format is the same as an S1 type record except for a 24-bit load address (6 hex characters versus 4). The format for an S2 type record is as follows:

```
S2 bb aaaaaa dd...dd xx
```

(spaces are inserted for clarity, each letter = one hex character)

Example: `S2 06 A00000 6C 01 EC`

(spaces added for clarity)

S3 Type

The S3 type is a data record starting on a 32-bit load address. The S3 type format is the same as an S1 type record except for a 32-bit load address (8 hex characters versus 4). The format for an S3 type record is as follows:

```
S3 bb aaaaaaaa dd...dd xx
```

(spaces are inserted for clarity, each letter = one hex character)

Example: `S3 07 A0000000 6C 01 EB`

(spaces added for clarity)

S5 Type

The S5 type contains the number of S1, S2, and S3 records which are in the record block. (A record block begins with an S0 record and ends with an S7, S8, or S9 record.) The format for an S5 type record is as follows:

```
S5 bb cccccccc xx
```

(spaces are inserted for clarity, each letter = one hex character)

Where:	bb	is the byte count
	cccccccc	is a count of the number of S1, S2, and S3 records in the block
	xx	Checksum = one's complement of: bb+cc+cc+cc+cc.

Example:

```
S5 05 00008000 F2
```

(spaces added for clarity)

S7 Type

The S7 type is an end record for a block of S3 records and has the following format.

S7 bb tttttttt xx

(spaces are inserted for clarity, each letter = one hex character)

Where:		
	bb	is the byte count
	tttttttt	is an optional 32-bit transfer address
	xx	Checksum = one's complement of: bb+tt+tt+tt+tt, or the one's complement of "bb" if the transfer address is not present.

Example: **S7 05 80008000 FA**

(spaces added for clarity)

S8 Type

The S8 type is an end record for a block of S2 records and has the following format.

`S8 bb tttttt xx`

(spaces are inserted for clarity, each letter = one hex character)

Where:		
	bb	is the byte count
	tttttt	is an optional 24-bit transfer address
	xx	Checksum = one's complement of: bb+tt+tt+tt, or the one's complement of "bb" if the transfer address is not present.

Example: **`S8 04 006600 95`**

(spaces added for clarity)

S9 Type

The S9 type is an end record for a block of S1 records and has the following format.

`S9 bb tttt xx`

(spaces are inserted for clarity, each letter = one hex character)

Where:

bb	is the byte count
tttt	is an optional 16-bit transfer address
xx	Checksum = one's complement of: bb+tt+tt, or the one's complement of "bb" if the transfer address is not present.

Example: **`S9 03 0000 FC`**

(spaces added for clarity)

T_hex Absolute File Format

Introduction

The Tektronix Hexadecimal (T_hex) File Format is used to transfer 8-bit processor absolute information. The definition of T_hex format includes a data transfer protocol in addition to specifying the absolute data representation. This protocol provides for positive and negative acknowledgement of records received and for retransmission of erroneous records.

The T_hex format specifies three types of records: data, terminating, and abort. The Tektronix format specifies that a maximum of 30 data bytes may be transmitted in any one data record. The record types are discussed in the following paragraphs.

Data Record

The format for a data record is as follows:

```
/ aaaa bb cc dddd...dd xx
```

(spaces are inserted for clarity, each letter represents one hex character)

Where:		
	/	record header character
	aaaa	4 hex digits specifying load address
	bb	byte count (# data bytes)
	cc	first checksum (cc = a+a+a+a+b+b)
	dd..dd	hex data
	xx	second checksum (data checksum).

Example of a Data Record:

```
/ 0A00 02 0C 2A C3 1B
```

(spaces are inserted for clarity)

Where:		
	0A00	load address (hex)
	02	data byte count
	0C	1st checksum = 0+A+0+0+0+2
	2A,C3	two data bytes
	1B	2nd checksum = 2+A+C+3.

Terminating Record

The format for a terminating record is as follows:

```
/ aaaa bb cc
```

(spaces are inserted for clarity)

Where:

/	record header character
aaaa	transfer address
bb	byte count (always 0)
cc	checksum (same as 1st checksum in data record).

Abort Record

The format for an abort record is as follows:

```
//text
```

An abort record is identified as having two header characters. The text is optional and may be used to describe the condition which caused the abort. The terminal does not send any text with an abort block. Only the device transmitting a file may send an abort. An acknowledgement is not expected after an abort record has been sent.

Extended T_hex Absolute File Format

Introduction

The extended T_hex file format is made up of three types of records: data records, symbol records, and termination records. Data and symbol records can be up to 255 characters long (not counting the end-of-line or carriage return).

Note

The **xlate** absolute file translator will not convert symbol records.

Variable-Length Fields

Certian fields of extended T_hex file format records can be variable lengths, i.e., from 2 to 17 characters in length. The first character of a variable-length field specifies the number of characters which follow (0 specifies that 16 characters follow in the particular field). For example, some variable-length symbols and values are: 4DONE, 3END, 5DELAY, 2FF, 3FFF, and 40000.

Valid Characters

Because symbols can be a part of the extended T_hex file format, characters other than hexadecimal characters can be a part of the file. The valid symbol characters are shown in table E-1 and equated with decimal values used for checksum computation. Data and termination records of the extended T_hex file format are described below. The character lengths of the various record fields, if known, are shown in parentheses after the field descriptions.

Table E-1. Extended T_hex Character Values

CHARACTER	DECIMAL VALUES
0 . . 9	0 . . 9
A . . Z	10 . . 35
$	36
%	37
. (period)	38
_ (underscore)	39
a . . z	40 . . 65

Data Record

Header Field (6)	Load Address (2 - 17)	Bytes of Hex Data

Load Address — This variable-length address specifies where the code is to be loaded.

Data Bytes — This part of the data record contains the object code.

Header Field

%	Record Length (2)	T	Checksum (2)

%

The "%" character specifies that the record is an extended T_hex record.

Record Length

This is the number of characters in the record (excluding the %) represented in two hex digits.

T

This character specifies the type of record where:

6 = Data Record.
3 = Symbol Record.
8 = Termination Record.

Checksum

This is a two digit hex number which equals the sum of all characters except the %, the checksum digits, and the end-of-line (or carriage return), mod 256.

Symbol Record

The **xlate** absolute file converter will not convert symbol records.

Header Field (6)	Section Name (2 - 17)	Section Def. Field *	Symb. Field	Symb. Field	

Header Field — See the previous description.

Section Name — This is a variable-length name of the section in which the symbols that follow (in the record) are found.

Section Definition Field

* The section definition field will appear in exactly one symbol record for each section.

0	Base Address (2 - 17)	Length (2 - 17)

Base Address — This variable-length address is the starting address of the section.

Length — This variable-length number equals 1 + (high address - base address).

Symbol Field

T	Symbol (2 - 17)	Value (2 - 17)

T — This character specifies the type of symbol where:

1 = Global Address.
2 = Global Scalar.
3 = Global Code Address.
4 = Global Data Address.
5 = Local Address.
6 = Local Scalar.
7 = Local Code Address.
8 = Local Data Address.

Symbol — A variable-length symbol.

Value — This is a variable-length value that is associated with the symbol.

Termination Record

Header Field (6)	Transfer Address (2 - 17)

Header Field — See the previous description.

Transfer Address — This is a variable-length address where program execution is to begin.

Hewlett-Packard Absolute File Format

Introduction

The absolute file consists of a variable number of records, the first of which, called the Processor Information Record, provides information about the microprocessor for which the file is intended. All subsequent records, called Data Records, are of variable length up to 128 sixteen-bit words and contain header information about the record along with data words.

For a pictorial representation of the absolute file format, see figures F-1, F-2, and F-3.

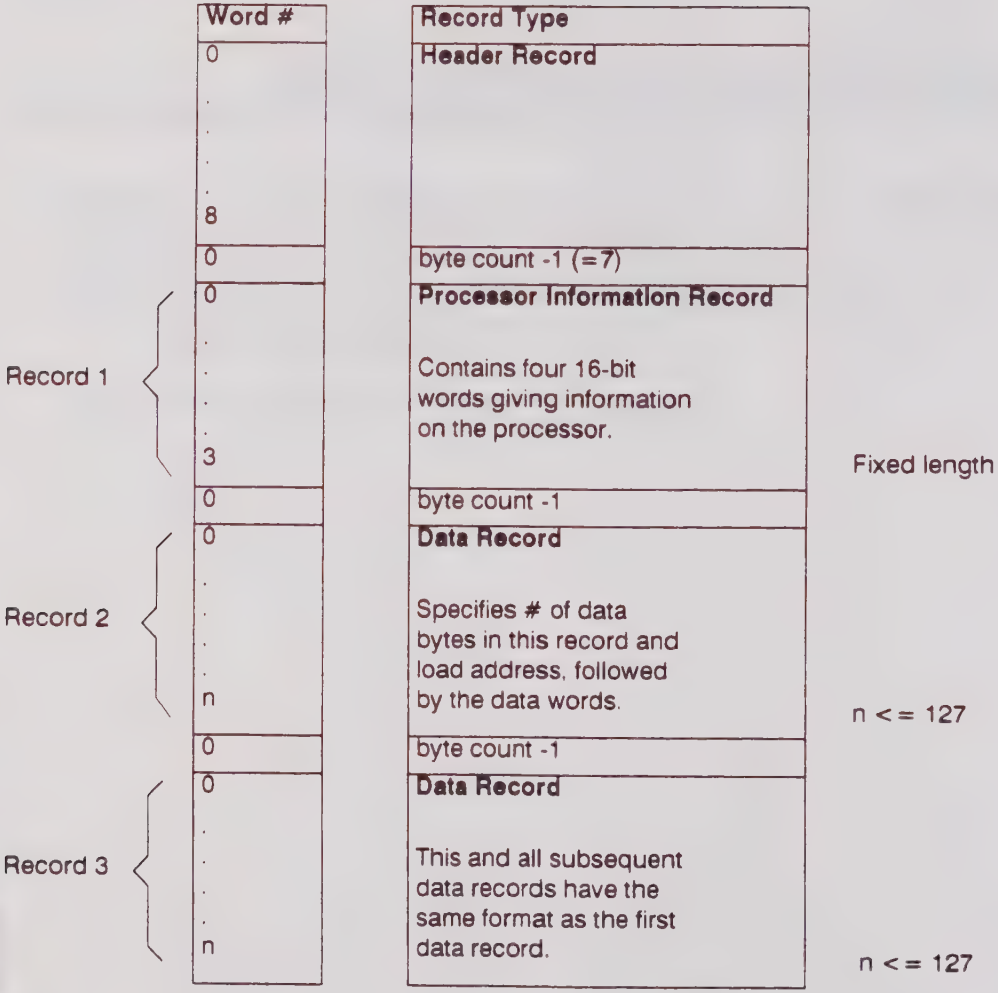


Figure F-1. HP Absolute Files - Overall Structure

Processor Information Record

WORD 0. Data Bus Width is the width of the processor data bus (8 or 16, for example).

WORD 1. Data Width Base is the minimum addressable entity (group of bits) used by the microprocessor. Usually this will be 8, but not always.

WORD 2-3. Transfer Address is the value to be loaded into the microprocessor Program Counter by the emulator. It is generated only by the linker and is set to zero when an absolute file is created by storing memory from the emulator. The Most Significant Word of the Transfer Address should be set to zero if it is not needed by the processor.

Word #	B15 B8	B7 B0
0	Data Bus Width	
1	Data Width Base	
2	Transfer Address LS Word	
3	Transfer Address MS Word	

Fixed Length = 4 words

Figure F-2. HP Absolute File - Processor Info. Record

Data Record

WORD 0. number of Data Bytes in this record expressed in binary.

WORD 1-2. Load Address is the binary address in the microprocessor memory space into which the first data byte (from WORD 3) should be loaded. Subsequent data from this record is loaded into the following microprocessor memory space. The Most Significant Word of the Load Address should be set to zero if it is not needed by the processor.

WORD 3 through n. Data Words are the binary representation of the absolute data to be loaded into microprocessor memory space.

WORD n. If the last byte of word n is not used, it should be set to 0.

Word #	B15 B8	B7 B0
0	# of Data Bytes in Record	
1	Load Address LS Word	
2	Load Address MS Word	
3	Data Word 1	
.	.	
.	.	
.	.	
n	Data Word m	

n <= 127

Figure F-3. HP Absolute File - Data Record Format

Index

E

error messages
  xlate, fatal errors **15**
  xlate, informational messages **14**
extended Tektronix hex format, valid characters **41**

F

fatal errors (xlate) **15**

H

header field **43**
HP absolute file format **47 - 50**
HP data record **50**
HP processor information record **49**

I

I_hex absolute file format **21 - 24**
informational messages (xlate) **14**
Intel 286_omf file format **25 - 28**

M

M_hex absolute file format **29 - 35**

O

option definitions
  xlate command line (HP-UX) **12**

P

prom programmer **9**

R

ROM, programming **9**

S

section definition field **44**
symbol field **45**
syntax
  xlate (HP-UX) **11**

T

T_hex absolute file format **37 - 40**

V

valid characters in extended Tektronix hex format **41**
variable length fields **41**

X

xlate (absolute file translator) **9**
  command line option definitions (HP-UX) **12**
  file formats **9**
  syntax (HP-UX) **11**

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse,

operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Hewlett-Packard
Printed in the USA

Local Area Sales Offices
Arranged alphabetically by country

Albania
IBS (Information & Business Systems)
Hewlett-Packard Distributor
1, Rr. Kongresi Permetit
Tirana
tel (5) 422385
fax (5) 4223864

Algeria
R.T.I.
Realisation Telematiques Internationales
Lot A, Villa no 57
Baba - Hassen
Wilaya de Tipaza
Algerie
tel (213) 230-6430
fax (213) 230-6431

Angola
Handled by HiPerformance, South Africa

Argentina
Hewlett-Packard Argentina
Montaneses 2140/50
1428 Buenos Aires
tel (54-1) 787-7115, 787-7287

Australia
Canberra
Hewlett-Packard Australia Ltd.
Thynne Street, Fern Hill Park
Bruce, ACT 2617
tel (61/6) 251-6999
fax (61/6) 251-6948

Melbourne
Hewlett-Packard Australia Ltd.
31-41 Joseph Street
Blackburn, VIC 3130
tel (61/3) 9272-2895
fax (61/3) 9898-7831

Perth
Measurement Innovation (WA) Pty. Ltd.
Kishorn Court
Suite 3, 58 Kishorn Road
Mount Pleasant WA 6153
tel (61/9) 316-2757
fax (61/9) 316-1392

Sydney
Hewlett-Packard Australia Ltd.
17-23 Talavera Road
North Ryde, N.S.W. 2113
tel (61/2) 950-7444
fax (61/2) 888-9072

Austria
Hewlett-Packard GmbH
Lieblgasse 1
A-1222 Vienna
tel (43/1) 25000-0
fax (43/1) 25000-444

Bahrain
BAMTEC
P.O.Box 10373
Capital Centre
Shop 116
Road 1405
Area 314
Manama
tel (973) 27 56 78
fax (973) 27 66 69

Belgium
Hewlett-Packard Belgium S.A./N.V.
Boulevard de la Woluwe
100/102 Woluwedal
B 1200 Brussels
tel (32/2) 778-34-17
fax (32/2) 778-34-14

Brazil
Hewlett-Packard Brasil
Alameda Rio Negro, 750
06454-000 - Alphaville - Barueri -
São Paulo
tel (55-11) 726-8155, 726-8160
fax (55-11) 726-8171

Hewlett-Packard Brasil
Rua Lauro Mueller, 116
Grupo 803 - Edificio Torre Rio Sul
22290-160 - Botafogo -
Rio de Janeiro
tel (55-21) 541-4404
fax (55-21) 295-2195

Brunei
Komputer Wisman Sdn Bhd
Room 1, 3rd Floor
Jalan Tutong
BS Begawan
Brunei Darussalem
tel (673/2) 23918
telex (0809) 2447

Bulgaria
S&T Bulgaria Ltd.
Hewlett-Packard Distributor
114 Vasil Levski Blvd.
1504 Sofia
tel (2) 468429
fax (2) 468391

Canada
Hewlett-Packard (Canada) Ltd.
5150 Spectrum Way
Mississauga, Ontario
L4W 5G1
tel (905) 206-4725
fax (905) 206-4739

Alberta
Hewlett-Packard (Canada) Ltd.
150 6th Avenue S. W.
Petro Canada Tower, Ste. 3600
Calgary, Alberta
T2P 3Y7
tel (403) 262-0777
fax (403) 237-9309

Hewlett-Packard (Canada) Ltd.
10155 102nd Street
Commerce Place, Suite 2100
Edmonton, Alberta
T5J 4G8
tel (403) 425-4725
fax (403) 420-4550

British Columbia
Hewlett-Packard (Canada) Ltd.
13575 Commerce Parkway, Ste. 110
Richmond, British Columbia
V6V 2L1
tel (604) 270-2277
fax (604) 270-0859

Manitoba
Hewlett-Packard (Canada) Ltd.
200 Graham Avenue, Ste. 1000
Winnipeg, Manitoba
R3C 4L5
tel (204) 942-4725
fax (204) 989-3553

New Brunswick
(see Nova Scotia)

Newfoundland
(see Nova Scotia)

Nova Scotia
Hewlett-Packard (Canada) Ltd.
238 Brownlow Avenue, Ste. 103
Park Place Centre
Dartmouth, Nova Scotia
B3B 1Y2
tel (902) 468-4725
fax (902) 468-4311

Ontario
London *(see below)*

Hewlett-Packard (Canada) Ltd.
5150 Spectrum Way
Mississauga, Ontario
L4W 5G1
tel (905) 206-4725
fax (905) 206-4739

Ottawa
Hewlett-Packard (Canada) Ltd.
2670 Queensview Drive
Ottawa, Ontario
K2B 8K1
tel (613) 820-6483
fax (613) 820-0377

Prince Edward Island
(see Nova Scotia)

SALES OFFICES
Arranged alphabetically by country (cont'd)

Quebec
Hewlett-Packard (Canada) Ltd.
17500 Trans Canada Highway
South Service Road
Kirkland, Quebec
H9J 2X8
tel (514) 697-4232
fax (514) 697-6941

Quebec City
(see Quebec)

Saskatchewan
(see Manitoba)

Chile
ASC. S.A.
Austria 2041
Santiago
tel (56-2) 274-7676
fax (56-2) 223-1912

China, People's Republic of
Beijing
China Hewlett-Packard Co., Ltd.
5-6/F West Wing Office
China World Trade Center
No. 1 Jian Guo Men Wei Avenue
Beijing 100004 - PRC
tel (86/10) 505-3888
fax (86/10) 505-0394
telex 085-22601 CTSHP CN
cable 1920 Beijing

Chengdu
China Hewlett-Packard Co., Ltd.
7/F, the Electronic Foreign Trade Building
#88 East 2nd Section,
Yi-Huan-Lu Road
Chengdu 610051, PRC
tel (86/28) 443-2666
fax (86/28) 443-2618

Guangzhou
China Hewlett-Packard Co., Ltd.
Room 1201-1204,
Gitic Plaza Main Bldg.
339 Huanshi Dong Road
Guangzhou 510060 - PRC
tel (86/20) 331-1421
fax (86/20) 331.1413
telex 44556 GPHGZ CN

Shanghai
China Hewlett-Packard Co., Ltd.
Room 638, Office Building
Shanghai Center
1376 Nan Jing Road West
Shanghai 200040 - PRC
tel (86/21) 279-8668
fax (86/21) 279-8943
telex 086-33577 CHPSB CN

Shenzhen
China Hewlett-Packard Co., Ltd.
4/F Huaqiang Component Building Zhenhua
Road, Shangbu District Shenzhen City
518031 - PRC
tel (86/755) 368-608
fax (86/755) 363-154

Xi'An Branch Office
3/F Dongxin Scientific
and Technical Trade Center
No 16A Middle Section
Yan-Ta-Lu Road Xi'An
PRC
tel (86/29) 526.9379
fax (86/29) 526.9243

Colombia
Instrumentacion Ltda
Calle 115 No. 11-A-10
Bogota
tel (57-1) 612-1313
fax (57-1) 612-0805

Costa Rica
I.S. de Costa Rica
Calle 25, Avenida 6 & 8, No. 648
San Jose
tel (506) 233-3722
fax (506) 255-3528

Cyprus
Telerexa Ltd.
P.O. Box 1152
Valentine House
8 Stassandrou St.
Nicosia
tel (3572) 44.56.28
fax (3572) 44.55.84
telex 5845 tlrx cy

Czech Republic
Hewlett-Packard Ceskoslovensko spo
Novodvorska 82
14200 Praha 4
tel 2/471 73 21
fax 2/474 32 93

Denmark
Hewlett-Packard A/S
Kongevejen 25
DK-3460 Birkerod
tel 45 45 99 10 00
fax 45 45 82 06 30

Hewlett-Packard A/S
Brenstrupgaardsvej 13
8200 Aarhus N
tel 45 86 78 30 00
fax 45 86 78 30 33
telex 37409 hpas dk

Dominican Republic
Esacomp, S.A.
Calle Seminario No. 55
Ens. Piantini
Santo Domingo
tel (809) 563-6350
fax (809) 565-0332

Ecuador
CYEDE Cia. Ltda.
Avenida Eloy Alfaro 1749
Quito
tel (593-2) 44-4224
fax (593-2) 44-4223

Egypt
ORASCOM
Onsi Sawiris & Co
P.O. Box 1191
160, 26th July Street
Aguza/Cairo
tel (202) 302-69-30
fax (202) 344-02-01
telex 92768 orscm un

Finland
Hewlett-Packard Oy
Piispankalliontie 17
02200 Espoo (Helsinki)
tel (358/0) 887 21
fax (358/0) 8872 2923

Hewlett-Packard Oy
Teknologiantie 6
90570 Oulu
tel (358/81) 5515 144
fax (358/81) 5515 145

France
Hewlett-Packard France
Test et Mesure
Z.A. de Courtaboeuf
1 Avenue du Canada
91947 Les Ulis
tel 69 82 60 60
fax 69 82 60 61

Germany
Headquarters
Hewlett-Packard GmbH
Herrenberger Strasse 130
71034 Böblingen
tel (49) 7031/14-0
fax (49) 7031/14-2999

Bad Homburg
Hewlett-Packard GmbH
Hewlett-Packard Strasse 1
61352 Bad Homburg
tel (49) 6172/16-0
fax (49) 6172/16-1309

Berlin
Hewlett-Packard GmbH
Lntzowplatz 15
10785 Berlin
tel (49) 30/25452-0
fax (49) 30/25452-162

Böblingen
Hewlett Packard GmbH
Schickardstrasse 2
71034 Böblingen
tel (49) 7031/14-0
fax (49) 7031/14-6429

Bonn
Hewlett-Packard GmbH
Friedrich-Ebert-Allee 26
53113 Bonn
tel (49) 228/23400-1
fax (49) 228/235-315

Hamburg
Hewlett-Packard GmbH
Überseering 16
22297 Hamburg
tel (49) 40/63808-0
fax (49) 40/63808-327

Hannover
Hewlett-Packard GmbH
Baumschulenallee 20-22
30625 Hannover
tel (49) 511/9577-0
fax (49) 511/9577-126

Karlsruhe
Hewlett-Packard GmbH
Hewlett-Packard-Str. 8
76337 Waldbronn (Karlsruhe)
tel (49) 7243/602-0
fax (49) 7243/602-212

Leipzig
Hewlett-Packard GmbH
Wehlitzer Strasse 2
04435 Schkeuditz
tel (49) 34204/82-0
fax (49) 34204/82-220

München (Munich)
Hewlett-Packard GmbH
Eschenstrasse 5
82024 Taufkirchen
tel (49) 89/61412-0
fax (49) 89/61412-300

Nürnberg
Hewlett-Packard GmbH
Emmericher Strasse 13
90411 Nürnberg
tel (49) 911/9524-0
fax (49) 911/9524-140

Ratingen
Hewlett-Packard GmbH
Berliner Strasse 111
40880 Ratingen
tel (49) 2102/905-0
fax (49) 2102/905-300

Great Britain
See United Kingdom

Greece
G-Systems
Hewlett-Packard Distributor
76, Ymittou Street
116 34 Athens
tel (30/1) 7264045
fax (30/1) 726 40 20

Hong Kong
Hewlett-Packard (HK) Ltd.
17-21/F Shell Tower
Times Square Matheson Street
Causeway Bay
Hong Kong
tel (852) 2599-7777
fax (852) 2506-9261

Hungary
Hewlett-Packard Magyarorszag Kft.
Erzsebet kiralyne utja 1/c
1146 Budapest
tel (36) 1 2524505
fax (36) 1 2527441

India
Bangalore
Hewlett-Packard India Pvt. Ltd.
Embassy Point
150 Infantry Road
Bangalore 560001
tel (91/80) 225-3024, 225-3039
fax (91/80) 225-6186
telex 845-2773 HP IN

Bombay
Hewlett-Packard India Pvt. Ltd.
Unit 1, - B Wing
Ground Floor, Neelam Center
Hind Cycle Road
Worli, Bombay 400 025
tel (91/22) 493-4722, 493-4723
fax (91/22) 493-7823
telex 11-76812 HPIB IN

Calcutta
Hewlett-Packard India Pvt. Ltd.
803 Central Plaza
2/7 Sarat Bose Road
Calcutta 700 020
tel (91/33) 748709
fax (91/33) 748571

Hyderabad
Hewlett-Packard India Pvt. Ltd.
5-9-13 Taramandal Complex
9th Floor, Saifabad
Hyderabad 500 004
tel (91/40) 232-679
fax (91/40) 220.012, 231.756
telex 425-2422 HPIL IN

New Delhi
Hewlett-Packard India Pvt. Ltd. Paharpur
Business Centre
21 Nehru Place
New Delhi 110 019
tel (91/11) 647-2311
fax (91-11) 646-1117
telex 31-71127 PBC IN

Indonesia
P.T. BERCA
Hardayaperkasa
Jl. Taman Tanah Abang III No. 31
Jakarta, 10160
tel (62/21) 385-5381
fax (62-21)-380-9166

Ireland
Hewlett-Packard Ireland Ltd.
Hewlett Packard House
Stradbrook Road
Blackrock, Co. Dublin
tel (353/1) 284-4633
fax (353/1) 284-5134
telex 30430

Iran
Evertech
11 bis, rue Toepffer
1206 Geneve
tel 4122/346 92 92
fax 4122/346 86 91
telex 421 787

Israel
Computation and Measurement
Systems (CMS) Ltd.
11, Hashlosha Street
Tel-Aviv 67060
tel (9723) 538-03-80
fax (9723) 537-50-55
telex 371234 HPCMS

Italy
For information on any product or service call: (39) 02 92 122 999

Hewlett-Packard Italiana S.p.A.
Via G. di Vittorio 9
20063 Cernusco s/N (Mi)
tel (39/2) 92-121
fax (39/2) 92-141-485

Easy-call Informazioni di vendita e supporto (39/2) 92 122 999

Napoli
Hewlett-Packard Italiana S.p.A.
Via Emanuele Gianturco 92 G
80146 NAPOLI (Na)
tel (39/81) 73-40-100
fax (39/81) 73-40-109

Roma Eur
Hewlett-Packard Italiana S.p.A.
Viale del Tintoretto, 200
00142 Roma Eur
tel (39/6) 54-831
fax (39/6) 54-01-661

Torino
Hewlett-Packard Italiana S.p.A.
Via Praglia, 15
10044 Pianezza (To)
tel (39/11) - 9685.1
fax (39/11) - 96 79 383

Japan
For presale assistance in product selection and other product or general information:
Hewlett-Packard Japan Ltd.
Test & Measurement
Measurement Assistance Center
tel 81-426-48-0722
fax 81-426-48-1073

Akita
Hewlett-Packard Japan Ltd.
Nihonseimei Akita Chuodori Bldg.
4-2-7 Nakadori, Akita-shi
Akita 010

Atsugi
Hewlett-Packard Japan Ltd.
9-32 Tamuracho, Atsugi-shi
Kanagawa 243

SALES OFFICES
Arranged alphabetically by country (cont'd)

Chiba
Hewlett-Packard Japan Ltd.
WBG Malibu East Bldg. 19F
2-6 Nakase
Mihama-ku, Chiba-shi
Chiba 261-71

Fukuoka
Hewlett-Packard Japan Ltd.
Daisan Hakata-Kaisei Bldg.
13-6 Hakataekiminami Hakata-ku,
Fukuoka-shi
Fukuoka 812

Hachioji
Hewlett-Packard Japan Ltd.
9-1 Takakura-cho, Hachioji-shi
Tokyo 192

Hamamatsu
Hewlett-Packard Japan Ltd.
219-21 Motoshiro-machi,
Hamamatsu-shi
Shizuoka 430

Hiroshima
Hewlett-Packard Japan Ltd.
Nissei- Heiwakoen Bldg.
3-25 Nakajima-cho Naka-ku
Hiroshima 730

Kawasaki
Hewlett-Packard Japan Ltd.
3-2-2 Sakado
Takatsu-ku, Kawasaki-shi
Kanagawa 213

Kobe
Hewlett-Packard Japan Ltd.
1-3-2 Muroya Nishi-ku, Kobe-shi
Hyogo 651-22

Kumagaya
Hewlett-Packard Japan Ltd.
Nissei-Kumagaya Bldg.
2-42 Miya-machi, Kumagaya-shi
Saitama 360

Kyoto
Hewlett-Packard Japan Ltd.
Maeda NS Bldg.
518 Akinono-machi Nijyo-Kudaru
Karasuma, Nakakyo-ku
Kyoto 604

Matsumoto
Hewlett-Packard Japan Ltd.
Forum Honjo Bldg
2-4-1 Honjo,
Matsumoto-shi
Nagano 390

Mito
Hewlett-Packard Japan Ltd.
Mito Mitsui Bldg.
1-4-73 Sannomaru, Mito-shi
Ibaraki 310

Nagoya
Hewlett-Packard Japan Ltd.
Nagoya Kokusai Center Bldg.
1-47-1 Nakono Nakamura-ku,
Nagoya-shi
Aichi 450

Omiya
Hewlett-Packard Japan Ltd.
Mitsui-seimei Omiya Miyamachi Bldg.
2-96-1 Miyamachi, Omiya-shi
Saitama 303

Osaka
Hewlett-Packard Japan Ltd.
Chuo Bldg.
5-4-20 Nishinakajima Yodogawa-ku
Osaka 532

Sapporo
Hewlett-Packard Japan Ltd.
Sumitomo- seimei Sapporo Chuo Bldg. 1-1-14
Nanjo- higashi Chuo-ku, Sapporo-shi
Hokkaido 060

Sendai
Hewlett-Packard Japan Ltd.
Yamaguchi Bldg.
1-1-31 Ichibancho Aoba-ku,
Sendai-shi
Miyagi 980

Takaido
Hewlett-Packard Japan Ltd.
3-8-20 Takaido-higashi,
Suginami-ku
Tokyo 168

Toyota
Hewlett-Packard Japan Ltd.
Nakane- Nissei Bldg.
4-25-18 Nishi-machi,
Toyota-shi
Aichi 471

Tsukuba
Hewlett-Packard Japan Ltd.
Tsukuba-Mitsui Bldg.
1-6-1 Takezono,Tsukuba-shi
Ibaraki 305

Utsunomiya
Hewlett-Packard Japan Ltd.
Chiyodaseimei-Utsunomiya Bldg.
2-3-1 Odori,Utsunomiya-shi
Tochigi 320

Jordan
Scientific & Medical Supplies Co.
P.O. Box 1387
Amman
tel (962/6) 62-49-07
fax (962/6) 62-82-58
telex 21456 sabco jo

Korea
Hewlett-Packard Korea Ltd.
HP Korea House
25-12, Yoido-dong
Youngdeung po-ku
Seoul, 150-010
tel (82/2) 769-0114
fax (82/2) 784-7084

Taegu
Hewlett-Packard Co. Ltd.
18/F Young Nam Tower B/D
111, Shin Cheon-dong, Dong-Gu
Taegu
tel (82/53) 754-2666
fax (82/53) 752-4696

Taejeon
Hewlett-Packard Co. Ltd.
3rd F, Hong In Tower Officetel
536-9, Bong Myung-dong,
Yusung-gu, Taejeon
tel (82/42) 828-6700-4
fax (82/42) 828-6709

Kuwait
Al-Khaldiya Electronics &
Electricals Equipment Co.
P.O. Box 830
Safat 13009
tel (965) 48 13 049
fax (965) 48 12 983

Lebanon
Computer Information Systems C.I.S.
Lebanon Street
Chammas Bldg.
P.O.Box 11-6274
Dora-Beirut
tel (961/1) 405-413
fax (961/1) 601.906
tel/fax (3579) 514-256 (Cyprus-based line)
telex 42309 chacis le

Malawi
Handled by HiPerformance,
South Africa

Malaysia
Hewlett-Packard Sales (M) SDN BHD
Ground Floor,
Wisma CycleCarri
288, Jalan Raja Laut
50350 Kuala Lumpur
tel (60/3) 298-6555
fax (60/3) 291-5495

Penang
Hewlett-Packard Sales (M) SDN BHD
6/F Wisma Penang Garden
42 Jalan Sultan Ahmad Shah
10050 Penang
tel (60/4) 376-532
fax (60/4) 371-027
telex MA 41111

Mauritus
Handled by HiPerformance,
South Africa

Mexico
Hewlett-Packard de Mexico, S.A.de C.V.
Prolongación Reforma No.700
Col. Lomas de Santa Fé
01210 Mexico D.F.
tel (52-5) 326-4392
fax (52-5) 326-4301

Morocco
SICOTEL
Complexe des Habous
Tour C, avenue des Far
Casablanca 01
tel (212) 231-22-70
fax (212) 231-65-45
telex 27604

Mozambique
Handled by HiPerformance, South Africa

Nambia
Handled by HiPerformance, South Africa

The Netherlands
Hewlett-Packard Nederland B.V.
Postbus 667
1180 AR Amstelveen
tel 020-547.6669
fax 020-547.7765

New Zealand
Hewlett-Packard (N.Z.) Ltd.
186-190 Willis Street
Wellington - New Zealand
tel (64/4) 382-0400
fax (64/4) 384-3380

Nigeria
Management Information Systems Co. Ltd.
3 Gerrard Road
P.O. Box 53386
Ikoyi, Lagos
tel (2341) 269.34.23
fax (2341) 269.34.19
telex 23582 DATEC NG

Norway
Hewlett-Packard Norge AS
Drammensveien 169
Postboks 60 Skøyen
N-0212 OSLO
Norway
tel (47) 22-73-56-00
fax (47) 22-73-56-19

Oman
IMTAC LLC
P.O. Box 51196
Mina al Fahal
Muscat
tel (968) 707-727, (968) 707-723
fax (968) 796-639, (968) 797-709

Pakistan
Mushko Electronics (PVT) Ltd.
68-W Sama Plaza
Blue Area, G-7
Islamabad 44000
tel (92/51) 828-459
fax (92/51) 816-190
telex 54001 Muski Pk

Mushko Electronics (PVT) Ltd.
Oosman Chambers
Abdullah Haroon Road
Karachi 74400
tel (92/21) 524-131
fax (92/21) 568-2298
telex 29094 MUSKO PK

Panama
Electronico Balboa, S.A.
Edif. El Dorado
Ave El Paiscal #30
Apartado 4929
Panama 5
tel (507) 236-0722
fax (507) 236-1820

Peru
Cia Electro Medica, S.A.
Los Flamencos 145, Of. 301-302
San Isidro, Lima 27
tel (51-14) 241-4325
fax (51-14) 242-3703

Philippines
Hewlett-Packard
Philippines Corporated
9th Floor, Rufino Pacific Towers,
6784 Ayala Avenue, Makati
Metro Manilla
tel (63-2) 894.1451
fax (63-2) 811.0635

Poland
Hewlett-Packard Polska
Ul. Newelska 6
01-447 Warszawa
tel 22/375065
fax 22/374783
telex 813919

Portugal
Hewlett-Packard Portugal S.A.
Rua Gregório Lopes, Lote 1732A
Torre de Sto. António
1400 Lisboa
tel (351/1) 301 7330
fax (351/1) 301 6332/3

Puerto Rico
Hewlett-Packard Puerto Rico (3175)
Centro Europa
1492 Ponce de Leon Avenue, Suite 203
Santurce, PR 00907
tel (809) 289-8900
fax (809) 289-8925

Qatar
Qatar Datamations Systems
Al Ahed Building, 5th floor
Al Muthof Street
Doha
tel 974/43.99.00
fax 974/43.21.54
telex 4833

Romania
S&T Romania SRL
Hewlett-Packard Distributor
Sitraco Centre
Blvd. Dimitries Cantemir 1
Etaj II - 210, Sector 4
Bucuresti
tel (1) 3237250
fax (1) 3237200

Russia/CIS
Hewlett-Packard Company
Pokrovski Blvd. 4/17 Kv.12
101000 Moscow
Russia
tel (7/095) 92 35 01
fax (7/095) 230-2611

Saudi Arabia
Modern Electronics Establishment
P.O. Box 2769
King Abdul Azia St.
Al-Khobar 31952
tel (9663) 894 65 48, 895 17 64
fax (9663) 894 46 58
telex 871106 HPMEEK SJ
cable ELECTA, AL-KHOBAR

Modern Electronics Establishment
P.O. Box 1228
Zaben Building, Tahliah St.
Jeddah - 21431
tel (966/2) 661-1447
fax (966/2) 661-0857
telex 602712 FARNAS SJ
cable ELECTA, JEDDAH

Modern Electronics Establishment
P.O. Box 22015
Prince Musad Ibn Abdulaziz St.
Riyadh 11495
tel (966/1) 476-3030
fax (966/1) 476-2570/476-7649
telex 402049 MEERYD SJ
cable ROUFKO

Singapore
Hewlett-Packard Singapore (Sales) Pte. Ltd.
150 Beach Road
#29-00 Gateway West
Singapore 0718
tel (65) 291-9088
fax (65) 296-7089
telex 087-34209 HPSGSO RS

Slovak Republic
High Performance Systems
Hewlett-Packard Distributor
Polianky 5
844 14 Bratislava
tel 7/760217
fax 7/763408

Slovenia, Croatia, Macedonia
Hermes-Plus
Hewlett-Packard Distributor
Celovska 73
61000 Ljubljana
Slovenia
tel (661) 159 33 22
fax (661) 558 597

South Africa
HiPerformance Systems (Pty.) Ltd.
P.O. Box 120, Howard Place
7450 Cape Town
tel (27/21) 685-7922
fax (27/21) 685-3785

HiPerformance Systems (Pty.) Ltd. Private
Bag, Wendywood
2144 Sandton
tel (27/11) 806-1000
fax (27/11) 806-1213

SALES OFFICES
Arranged alphabetically by country (cont'd)

Spain
Barcelona
Hewlett-Packard Española, S.A.
Avda. Diagonal, 605
08028 Barcelona
tel 93-401 91 00
fax 93-430 84 68

Madrid
Hewlett-Packard Española, S.A.
Crta N-VI km 16,500
28230 Las Rozas, Madrid
tel 1-631 16 00
fax 1-631 18 30

Sevilla
Hewlett-Packard Española, S.A.
luis de Morales, 32
EdificoForum
41018 Sevilla
tel 95-453 29 10
fax 95-453 30 92

Valencia
Hewlett-Packard Española, S.A.
plaza de America, 2 - 5Y
46004 Valencia
tel 96-395 74 75
fax 96-395 42 11

Vizcaya
Avda de Zugazarte
Helett-Packard Española, S.A.
Avda de zugazarte, 8
Edif El Abra, 4
48930 Las Arenas, Guecho
tel 94-481 80 00
fax 94-481 80 40

Sri Lanka
Precision Technical Services (PVT)Ltd.
65/14, Park Street
Colombo 2
tel (941) 687791
fax (941) 687731

Sweden
Hewlett-Packard Sverige AB
Skalholtsgatan 9
164 97 Kista
tel (46/8) 444 20 00
fax (46/8) 444 25 25

Göteborg
Hewlett-Packard Sverige AB
Vädursgatan 6
Box 5328
402 27 Göteborg
tel (46/31) 35-18-00
fax (46/31) 35-18-99

Malmö
Hewlett-Packard Sverige AB
Hjälmaregatan 3
Box 371
201 23 Malmö
tel (46/40) 660 72 00
fax (46/40) 660 72 99

Malm
Hewlett-Packard Sverige AB
J gersrov gen 160
Box 28066
200 28 Malm
tel (46/40) 22-79-00
fax (46/40) 22-79-99

Service Office
Hewlett-Packard Sverige AB
Arabygatan
Box 3161
350 43 Växjö
tel (46/470) 396-06
fax (46/470) 154-05

Switzerland
Berne
Hewlett-Packard (Schweiz) AG
Meriedweg 11
CH-3172 Niederwangen (Bern)
tel (41) 31-980 31 11
fax (41) 31-980 33 90

Gèneva
Hewlett-Packard (Suisse) SA
39, rue de Veyrot
CH-1217 Meyrin 1
tel (41) 22-780 41 11
fax (41) 22-780 42 20

Zürich
Hewlett-Packard (Schweiz) AG
In der Luberzen 29
CH-8902 Urdorf/Zürich
tel (41) 1-735 71 11
fax (41) 1-735 77 00

Taiwan, ROC
Hewlett-Packard Taiwan Ltd.
Hewlett-Packard Building
8/F No. 337, Fu Hsing North Road
Taipei, 10483
tel (886/2) 712-0404
fax (886/2) 718-9860

Chungli
Hewlett-Packard Taiwan Ltd.
No. 20 Kao-Shuang Road
Ping-Chen, 32404
Taoyuan Hsieh
tel (886/3) 492-9666
fax (886/3) 492-9669

Kaohsiung
Hewlett-Packard Taiwan Ltd.
12/F,10, Su Wei 4 Road
Lin-Ya Distric
Kaohsiung
tel (886/7) 330-1199
fax (886/7) 330-8568

Taichung
Hewlett-Packard Taiwan Ltd.
10F, 540, Sec. 1
Wen-Hsin Road
Taichung
tel (886/4) 327-0153
fax (886/4) 326-5274

Thailand
Hewlett-Packard (Thailand) Ltd.
11/F Pacific Place
140 Sukhumvit Road
Bangkok 10110
tel (66/2) 254-6720
fax (66/2) 254-6731

Trinidad and Tobago
Borde Communications
1 Valsayn Road
Curepe, P.O. Box 897
Port-of-Spain, Trinidad
tel (809) 663-2087
fax (809) 645-3352

Tunisia
Precision Electronique
5, rue de Chypre
Mutuelle Ville
1002 Tunis Belvedere
tel (216/1) 785-037, 890-672
fax (216/1) 780-241
telex (0409) 18238 TELEC TN

Turkey
Hewlett-Packard
Bilgisayar ve Olcum
Sistemleri A.S. Karum
Business Center, Suite 461
Iran Caddesi 21
06680 Kavaklidere
Ankara
tel (90/312) 468 87 70
fax (90/312) 468 87 78

Ukraine
S&T Ukraine Ltd
50, Popudrenk St.
253660 Kiev
tel (44) 559 47 63, 559 29 88
fax (44) 559 50 33

United Arab Emirates
Abu Dhabi
Emitac Ltd.
P.O. Box 2711
Abu Dhabi
tel (971/2) 77-04-19
fax (971/2) 72-30-58
telex 46306

Dubai
Emitac Ltd.
Block RBS Arenco Bldg.
Zabel Road
P.O. Box 8391
Dubai
tel (971/4) 37-75-91
fax (971/4) 37-08-99
telex 48710

United Kingdom
Bracknell
Hewlett-Packard Limited
Cain Road
Bracknell
Berkshire RG12 1HN
tel (44/1344) 366666
fax (44/1344) 362852
telex 849 555

United States

To request formal quotations, place orders, and other order information:

Test & Measurement Customer
Business Center
P.O. Box 4026
Englewood, CO 80155-4026
tel 1-800-829-4444

For presale assistance in product selection and other product or general information:

Test & Measurement Customer
Information Center
tel 1-800-452-4844

Alabama

Hewlett-Packard Co. (3114)
620 Discovery Drive
Huntsville, AL 35806
tel (205) 971-2000
fax (205) 971-8899

Arizona

Hewlett-Packard Co. (2401)
8080 Pointe Parkway West
Phoenix, AZ 85044
tel (602) 273-8000
fax (602) 273-8022, 273-8080

Hewlett-Packard Co. (2406)
3400 East Britannia Drive
Bldg. C, Suite 130
Tucson, AZ 85706
tel (602) 573-7400
fax (602) 573-7429

California

Hewlett-Packard Co. (2403)
1421 S. Manhattan Avenue
Fullerton, CA 926301
tel (714) 999-6700
fax (714) 758-7559

Hewlett-Packard Co. (2438)
130 Cremona, Suite A
Goleta, CA 93117
tel (805) 685-6100
fax (805) 685-6163

Irvine *(see Fullerton)*

Hewlett-Packard Co. (2417)
5245 Pacific Concourse Drive, Suite 100
Los Angeles, CA 90045
tel (310) 643-5112
fax (310) 535-2601

Hewlett-Packard Co. (2426)
351 E. Evelyn Avenue
Bldg. 333
Mountain View, CA 94039
tel (415) 694-2000
fax (415) 694-3594

Hewlett-Packard Co. (2415)
3831 N. Freeway Blvd., Bldg.C, Suite 100
Sacramento, CA 95834
tel (916) 567-8500
fax (916) 567-8777

Hewlett-Packard Co. (2404)
9606 Aero Drive
San Diego, CA 92123
tel (619) 279-3200
fax (619) 268-8487, 279-0560

Santa Barbara *(see Goleta)*

Hewlett-Packard Co. (2428)
5805 Sepulveda Blvd, Suite 800
Van Nuys, CA 91411
tel (818) 786-5800
fax (818) 779-2350

Ventura *(see Van Nuys)*

Colorado

Hewlett-Packard Co. (2441)
3005 Center Green Drive, Suite 205
Boulder, CO 80301
tel (303) 938-3065
fax (303) 938-3025

Hewlett-Packard Co. (2411)
24 Inverness Place East
Englewood, CO 80112
tel (303) 649-5000
fax (303) 649-5787

Connecticut

Hewlett-Packard Co. (4457)
263 Tresser Blvd., 9th Floor
Stamford, CT 06901
tel (203) 324-1003
fax (203) 325-5680

Hewlett-Packard Co. (4458)
115 Glastonbury Blvd.
Glastonbury, CT 06033
tel (203) 633-8100
fax (203) 659-6087

District of Columbia
(see Rockville, MD)

Florida

Hewlett-Packard Co. (3179)
5900 N. Andrews, Suite 100
Fort Lauderdale, FL 33309
tel (305) 938-9800
fax (305) 938-2293

Hewlett-Packard Co. (3177)
6177 Lake Ellenor Drive
Orlando, FL 32809
tel (407) 859-2900
fax (407) 826-9309

Hewlett-Packard Co. (3166)
5550 Idlewild, Suite 150
Tampa, FL 33634
tel (407) 859-2900
fax (407) 826-9295

Georgia

Hewlett-Packard Co. (3108)
20 Perimeter Summit Blvd.
Atlanta, GA 30319
tel (404) 648-0000
fax (404) 648-7010

Hawaii

Hewlett-Packard Co. (2409)
3660 Waialae Ave, Suite 102
Honolulu, HI 96816
tel (808) 732-1566
fax (808) 739-3100

Illinois

Hewlett-Packard Co. (2619)
2205 E. Empire Street
Bloomington, IL 61704
tel (309) 664-4000
fax (309) 664-4100

Hewlett-Packard Co. (2603)
1200 East Diehl Road
Naperville, IL 60566
tel (708) 245-3000
fax (708) 245-3600

Hewlett-Packard Co. (2601)
5201 Tollview Drive
Rolling Meadows, IL 60008
tel (708) 342-2000
fax (708) 342-2022

Indiana

Hewlett-Packard Co. (2605)
201 W. 103rd Street, Suite 100
Indianapolis, IN 46290
tel (317) 582-4400
fax (317) 582-4500

Hewlett-Packard Co. (2635)
111 E. Ludwig Road, Suite 108
Ft. Wayne, IN 46825
tel (219) 480-3700
fax (219) 480-3730

Iowa

Hewlett-Packard Co. (2612)
4050 River Center Court
Cedar Rapids, IA 52402
tel (319) 393-0606
fax (319) 378-1024

Kansas

Hewlett-Packard Co. (2622)
P.O. Box 780408
3450 N. Rock Road, Suite 300
Wichita, KS 67226
tel (316) 636-4040
fax (316) 636-4966

Kentucky

Hewlett-Packard Co. (2623)
305 N. Hurstbourne Parkway
Suite 100
Louisville, KY 40222-5141
tel (502) 329-4800
fax (502) 329-4890

Maryland

Hewlett-Packard Co. (4432)
3701 Koppers Street
Baltimore, MD 21227
tel (410) 644-5800
fax (410) 362-7650

Hewlett-Packard Co. (4430)
2101 Gaither Road
Rockville, MD 20850
tel (301) 258-2000
fax (301) 258-5986

Massachusetts

Hewlett-Packard Co. (4450)
29 Burlington Mall Road
Burlington, MA 01803-4182
tel (617) 270-7000
fax (617) 221-5240

SALES OFFICES
Arranged alphabetically by country (cont'd)

Michigan

Hewlett-Packard Co. (2617)
3033 Orchard Vista S.E., Suite 100
Grand Rapids, MI 49546
tel (616) 975-3200
fax (616) 975-3290

Hewlett-Packard Co. (2607)
39550 Orchard Hill Place Drive
Novi, MI 48376
tel (810) 380-2100
fax (810) 380-2450

Minnesota

Hewlett-Packard Co. (2606)
2025 W. Larpenteur Avenue
St. Paul, MN 55113
tel (612) 644-1100
fax (612) 641-9787

Missouri

Hewlett-Packard Co. (2611)
6601 Winchester Avenue
P.O. Box 18230
Kansas City, MO 64133-8230
tel (816) 737-0071
fax (816) 737-4690

Hewlett-Packard Co. (2610)
530 Maryville Centre Drive
St. Louis, MO 63141
tel (314) 542-1500
fax (314) 542-1585

Nebraska

Hewlett-Packard Co. (2616)
2533 N. 117th Avenue
Omaha, NE 68164
tel (402) 493-0300
fax (402) 491-6230

New Jersey

Hewlett-Packard Co. (4415)
West 120 Century Road
Paramus, NJ 07653
tel (201) 599-5000
fax (201) 599-5629

Hewlett-Packard Co. (4417)
20 New England Ave. West
Piscataway, NJ 08854
tel (908) 562-6100
fax (908) 562-6406

New Mexico

Hewlett-Packard Co. (2402)
5130 Masthead NE
Albuquerque, NM 87109
tel (505) 823-6100
fax (505) 823-1243

Hewlett-Packard Co. (2420)
1362-C Trinity Drive
Los Alamos, NM 87544
tel (505) 662-6700
fax (505) 662-4312

New York

Hewlett-Packard Co. (4444)
114 Great Oaks Office Park
Albany, NY 12203
tel (518) 452-9947
fax (518) 452-8989

Buffalo *(see Cheektowaga)*

Hewlett-Packard Co. (4445)
Airport Commerce Park
305 Cayuga Road, Suite 160
Cheektowaga, NY 14225
tel (716) 264-4100
fax (716) 264-4150

Hewlett-Packard Co. (4440)
5010 Campuswood Drive
East Syracuse, NY 13057
tel (315) 463-2727
fax (315) 463-3150

Hewlett-Packard Co. (4442)
290 Woodcliff Drive
Fairport, NY 14450
tel (716) 264-4000
fax (716) 264-4150

Huntington *(see Melville)*

Hewlett-Packard Co. (4411)
7 Corporate Center Drive
Melville, NY 11747
tel (516) 753-0555
fax (516) 753-3438

Hewlett-Packard Co. (4412)
399 Park Avenue, 26th Floor
New York, NY 10022
tel (212) 350-6700
fax (212) 350-6810

Rochester *(see Fairport)*

Hewlett-Packard Co. (4409)
Executive Square Office Bldg.
66 Middlebush Road
Wappinger Falls, NY 12590
tel (914) 298-9125
fax (914) 298-3338

North Carolina

Hewlett-Packard Co. (3142)
2000 Regency Parkway, Suite 175
Cary, NC 27511
tel (919) 467-6600
fax (919) 460-2296

Hewlett-Packard Co. (3145)
9401 Arrowpoint Blvd.
Charlotte, NC 28210
tel (704) 527-8780
fax (704) 559-7444

Hewlett-Packard Co. (3111)
7029 Albert Pick Road, Suite 100
Greensboro, NC 27409
tel (910) 665-1800
fax (910) 668-1797

Raleigh *(see Cary)*

Ohio

Hewlett-Packard Co. (2624)
4501 Erskine Road
Cincinnati, OH 45242
tel (513) 985-6200
fax (513) 985-6300

Cleveland *(see Strongsville)*

Hewlett-Packard Co. (2614)
540 Office Center Place
Columbus, OH 43230-5321
tel (614) 478-6200
fax (614) 478-6299

Hewlett-Packard Co. (2604)
7887 Washington Village Drive
Dayton, OH 45459
tel (513) 291-7500
fax (513) 291-7600

Hewlett-Packard Co. (2608)
15885 Sprague Road
Strongsville, OH 44136
tel (216) 243-7300
fax (216) 234-7230

Oklahoma

Hewlett-Packard Co. (3182)
3525 NW 56th Street Suite C100
Oklahoma City, OK 73112
tel (405) 946-9499
fax (405) 948-4760

Hewlett-Packard Co.(3183)
6655 South Lewis, Suite 105
Tulsa, OK 74136
tel (918) 481-2285
fax (918) 481-2250

Oregon

Hewlett-Packard Co. (2413)
15115 S.W. Sequoia Parkway
Suite 100
Portland, OR 97224
tel (503) 598-8000
fax (503) 598-8155

Pennsylvania

Hewlett-Packard Co. (2609)
111 Zeta Drive
Pittsburgh, PA 15238
tel (412) 782-0400
fax (412) 784-3340

Hewlett-Packard Co. (4420)
2750 Monroe Blvd.
Valley Forge, PA 19482-0713
tel (215) 666-9000
fax (215) 666-2034

Hewlett-Packard Co. (4420)
1400 Morris Drive, Suite 300
Wayne, PA 19087-5580
tel (610) 408-6000
fax (610) 408-6034

Tennessee

Hewlett-Packard Co. (3117)
404 BNA Drive
Bldg. 200, Suite 510
Nashville, TN 37217
tel (615) 367-4700
fax (615) 399-4616

Texas

Hewlett-Packard Co. (3197)
10415 III Morado Circle, Suite 100
Austin, TX 78759
tel (512) 346-3855
fax (512) 338-7201

Dallas *(see Richardson)*

Hewlett-Packard Co. (2405)
5700 Cromo Drive, Suite 100
El Paso, TX 79912
tel (915) 833-4400
fax (915) 833-4444

Hewlett-Packard Co. (3187)
2000 West Loop South
Houston, TX 77027
tel (713) 439-5300
fax (713) 439-5495

Hewlett-Packard Co. (3185)
930 E. Campbell Road
Richardson, TX 75081
tel (214) 231-6101
fax (214) 699-4337

Hewlett-Packard Co. (3188)
14100 San Pedro Avenue, Suite 100
San Antonio, TX 78232
tel (210) 494-9336
fax (210) 491-1299

Utah

Hewlett-Packard Co. (2412)
348 E. Winchester, Suite 150
Salt Lake City, UT 84107
tel (801) 265-6200
fax (801) 265-6390

Virginia

Chesapeake *(see Glen Allen)*

Hewlett-Packard Co. (4435)
3141 Fairview Park Drive, Suite 300
Falls Church, VA 22042
tel (703) 204-2100
fax (703) 204-2425

Hewlett-Packard Co. (3113)
4401 Waterfront Drive, Suite 150
Glen Allen, VA 23060
tel (804) 747-7750
fax (804) 965-9297

Norfolk *(see Glen Allen)*

Richmond *(see Glen Allen)*

Roanoke *(see Glen Allen)*

Washington

Hewlett-Packard Co. (2410)
15815 S.E. 37th Street
Bellevue, WA 98006
tel (206) 643-4000
fax (206) 643-8748

Wisconsin

Hewlett-Packard Co. (2615)
250 North Patrick Blvd., Suite 100
Brookfield, WI 53045
tel (414) 879-2200
fax (414) 879-2218

Uruguay

CONATEL S.A.
Ejido 1690
Montevideo
tel (598-2) 92-0314, 92-1276
fax (598-2) 92-0660

Venezuela

Hewlett-Packard de Venezuela
3ra. Transversal, Los Ruices Norte
Edificio Segre, Caracas
tel (58-2) 239-4133
fax (58-2) 239-3080

Zambia

Handled by HiPerformance, South Africa

Zimbabwe

Handled by HiPerformance, South Africa

Please Send Directory Corrections to:

Test & Measurement Catalog
MS: 54L-AF
Hewlett-Packard Company
5301 Stevens Creek Blvd.
P.O. Box 58059
Santa Clara, CA 95052-8059
tel (408) 553-7271
fax (408) 553-6779

Regional Sales Offices

If no T&M sales office is listed for your country or area, please contact one of these main regional offices:

Asia Pacific

Hewlett-Packard Asia Pacific Ltd.
17-21/F Shell Tower, Times Square
1 Matheson Street, Causeway Bay
Hong Kong
tel (852) 2599-7777
fax (852) 2506-4997

Hewlett-Packard Japan Ltd.
3-29-21 Takaido-higashi, Suginami-ku
Tokyo 168
Japan
tel (81/3) 3331-6111
fax (81/3) 3335-1478

Canada

Hewlett-Packard (Canada) Ltd.
5150 Spectrum Way
Mississauga, Ontario L4W 5G1
tel (905) 206-4725
fax (905) 206-4739
telex 419019 hps

Eastern Central Europe

Hewlett-Packard GmbH
Lieblgasse 1
P.O. Box 72
A-1222 Vienna, Austria
tel (43/1) 25000-0
fax (43/1) 25000-444
telex 2134-425

Europe

Hewlett-Packard S.A.
Route du Nant-d'Avril 150
CH-1217 Meyrin 2
Switzerland
tel (41) 22/780-81-11
fax (41) 22/780-85-42
telex 419019 hps

Latin America

Hewlett-Packard Co.
5200 Blue Lagoon Drive, 9th Floor
Miami, Florida, U.S.A. 33126
tel (305) 267-4245
fax (305) 267-4288

Middle East and Africa

Hewlett-Packard S.A.
International Sales Branch
Middle East/Africa
Rue de Veyrot 39
P.O. Box 364
1217 Meyrin 1
Gèneva, Switzerland
tel (41/22) 780-4111
fax (41/22) 780-4770
telex 418907

United States

P.O. Box 4026
Englewood, CO 80155-4026
tel 1-800-452-4844

5959-6047

Printed in U.S.A.